2018-19 SENIOR HONORS THESIS

ISHAN SHAH

SUBJECT: EXPOSITORY DEVELOPMENT OF MATERIALS FOR LIVE ONLINE INSTRUCTION OF *STATISTICS FOR RISK MODELING*

ADVISOR: DR. ROBIN CUNNINGHAM, UNC DEPARTMENT OF STATISTICS AND OPERATIONS RESEARCH

EXECUTIVE SUMMARY

Throughout the year, I developed materials to aid students' understanding of the statistical learning component in the future UNC course STOR 518W: "Statistics for Risk Modeling" (SRM). The primary course instructor and designer is my advisor, Dr. Robin Cunningham. This course is designed for the Society of Actuaries actuarial exam of the same name. The primary text for my portion of the course is *An Introduction to Statistical Learning, with Applications in R* (*ISLR*) by James, Witten, Hastie, and Tibshirani.

I was responsible for the notes from inception and used RMarkdown along with LaTex as my primary working environment. The result is "slidy" presentations that are internet-friendly and summarize concepts and formulas from the relevant chapters of *ISLR*. Using a two-camera, office studio, I created and edited videos with the software packages Explain Everything and iMovie (see links in the main report) in which I detail the solutions of relevant practice questions, both theoretical and applied. Videos include both written solutions along with simultaneous video of me solving them to increase student engagement. This included questions are from the end of *ISLR* Chapters and from Society of Actuaries materials. I used *R* as my primary programming language for the development of applied examples. All of the videos and notes will be used in the pilot semester of the new course STOR 518W.

For my work, I have occasionally have drawn inspiration from Dr. Yufeng Liu's Spring 2018 STOR 565 (Machine Learning) lectures. Also, learning and using RMarkdown proved a significant improvement over PowerPoint slides in that one can easily embed LaTeX for formulas as well as display *R* code and its output, without sacrificing essential slideshow tools such as text and images.

The *ISLR* chapters and material covered include:

Chapter 2 (Notes, Video Created): Overview of Statistical Learning (Regression vs. Classification, Unsupervised vs. Supervised learning, Bias-Variance tradeoff), K-Nearest Neighbors

Chapter 3 (Notes, Video Created): Simple Linear Regression and Multiple Linear Regression from a Statistical Learning perspective

Chapter 5 (Notes, Written Example Created): Different types of Cross-Validation

Chapter 6 (Notes, Written Example, Video Created): Linear model selection through best subsets, stepwise methods, and shrinkage methods (Ridge/LASSO regression)

Chapter 8 (Notes, Written Example Created): Decision (Regression/Classification) Trees, Bagging, Boosting, Random Forests

Chapter 10 (2 Notes Created, Video Created): Unsupervised learning through Principal Component Analysis and clustering methods (hierarchical, k-means)

Other tasks I have completed include writing a review comparing *ISLR* and another textbook *Actuarial Statistics with R* by Gan and Valdez. I concluded that *ISLR* is best to use as the primary resource due to its coherent incorporation of material, formulas, and examples. The other text is more suited as a secondary reference as it mainly lists formulas without sufficient context.

Below are the links to the completed instructional videos, followed by the book review and the slideshows I have created in RMarkdown (notes and applied examples).

LINKS TO COMPLETED VIDEOS

All videos are posted on my personal YouTube channel and are unlisted, meaning it can only be reached if one has a link to the video.

Chapter 2 (Overview of Statistical Learning):

www.tinyurl.com/SRM-ISLR-CH2

Chapter 3 (Linear Regression):

www.tinyurl.com/SRM-ISLR-CH3

Chapter 6 (Model Selection and Shrinkage Methods):

www.tinyurl.com/SRM-ISLR-CH6

Chapter 10 (Clustering Methods):

www.tinyurl.com/SRM-ISLR-CH10

BOOK REVIEW: *ISLR vs. AS*

Thoughts after looking at Chapter 3 in both related to regression (specifically multi-linear regression, an important topic):

Based on this chapter, **I think *ISLR* would be a better primary textbook, with *Actuarial Statistics (AS)* good to use as a reference after the initial material is taught**. Here's why:

Main Chapters: Both textbooks seem to have the necessary formulas laid out to readers. In fact, *AS* definitely has them in a very organized, sequential manner, whereas in *ISLR* it is scattered throughout text chunks. In the case where one wants to simply look up a formula quickly, *AS* may be the better resource. However, the *AS* non-Case Study chapters do not provide any examples nor any visualizations. This book could easily confuse students who are entirely new to the subject and do not have a solid mathematics (linear algebra specifically) background. In contrast, *ISLR* has clear visualizations and uses examples continually to demystify complicated concepts - the cost being formulas not in a clear order. Also, for specifically regression, *ISLR* frames the topic more from a statistical learning perspective (training vs. test error, etc.) whereas *AS* has it in a more traditional, statistical manner (detailed matrix equations). The examples at the end of the *AS* chapter were essentially linear algebra problems and hardly relevant, where as *ISLR* has a good mix of conceptual and applied problems related to statistical learning and covers most of what was taught in the chapter.

Case Studies: Since the course is geared more towards actuarial students, the case studies in *AS* may be more relevant than the examples provided in *ISLR*. However, much of the example code is brute-force coding algorithms such as forward/backward selection and applying them to datasets. While this is important, there are many functions from R libraries that do these commands in a few, simple lines, which may be more worth it for a class aiming to teach statistics. *ISLR's* case study code seems a lot more concise and easier to follow.

PCA (*AS* Ch 14, *ISLR* Ch 10)

Similar to the regression chapter, *AS* describes principal components without much context and dives straight into the formulas, which are very linear algebra heavy due to the nature of PCA. It shows a very clear, straightforward way to define PCA and then compute them, but does little to show context as to what it actually means, making it hard for someone new to understand the concept. Again, *ISLR* puts PCA into context significantly better with examples from datasets and visualizations to show the principal component representation of the data. I believe that those reading *ISLR's* section, although longer than *AS*, will come with a better understanding of PCA's applications in unsupervised learning. Neither of the two books seem to have a good example of calculating the PCA's from start to finish though (we had a good one in STOR 565, do not remember where it came from).

Overall:

ISLR

- Longer, formulas less organized
- Much better visualizations and contextualizing the concepts
- Examples with more relevance to statistical learning
- Code more modern and concise
- **Good main textbook, enough examples for good understanding**

AS

- Better for actuarial examples (case studies), long-winded code
- Formulas organized sequentially, in-depth linear algebra
- No visualizations and context within main text - harder to understand
- Examples more lin-alg, traditional stats heavy
- **Good formula reference at best, good actuarial examples**

Chapter 2: Statistical Learning

Ishan Shah

September 8, 2018

What is Statistical Learning?

Predictors, Input Variables, Features $X_1, X_2, X_3, \ldots X_p$ are related to response variable Y through a function written in the general form: $Y = f(X) + \epsilon$

Statistical learning: the approach to finding the f that defines the relationship between X and Y

ϵ: Random error term, independent of X and has approximately mean zero

Uses: 1. To **predict** the response for future observations 2. To make **inferences** about predictor variables that go beyond the sample itself

Prediction

Based on a person's genetic markers, what is the risk they have an adverse reaction to a drug?

What will be the response to a new restaurant based on the demographics of the location?

Modeled by: $\hat{Y} = \hat{f}\left(X_1, X_2, \ldots, X_P\right)$

Goal: Minimize $E(Y - \hat{Y})^2$
$E(Y - \hat{Y})^2 = E[f(X) - \hat{f}\left(X\right)]^2 + Var(\epsilon)$

Can only minimize the *Reducible Error*: $E[f(X) - \hat{f}\left(X\right)]^2$

Irreducible Error: $Var(\epsilon)$

Inference

Involves using evidence from statistical models to make judgments about predictors and their specific relationship to the response variable - easier through models such as linear and logistic regression

Example questions: Having which genetic markers increases the chances of an adverse reaction the most?

How much will the restaurant's sales increase with a higher proportion of younger people?

Parametric Methods to Estimate f

Key: Already making an assumption about the model's functional form

Example: Linear Models modeled by

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$

Logistic Regression, Linear Discriminant Analysis

Advantages: Use methods that are not computationally expensive like **least squres** to find parameters $\beta_1, \beta_2, \ldots \beta_p$ instead of arbitrary $f(X)$, which leads to better interpretability of predictors

Disadvantages: Can be far from the true f, creating poor estimates

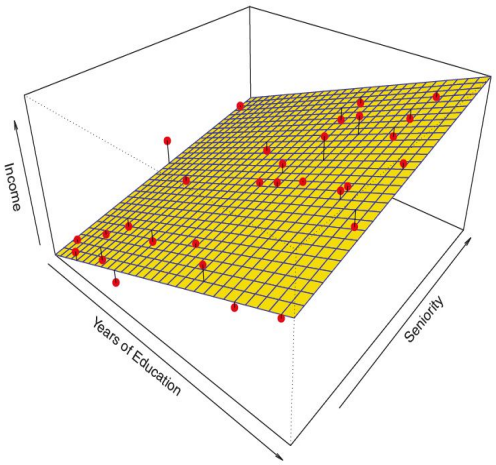


FIGURE 2.4. *A linear model fit by least squares to the* Income *data from Figure 2.3. The observations are shown in red, and the yellow plane indicates the least squares fit to the data.*

Non-parametric methods to estimate f

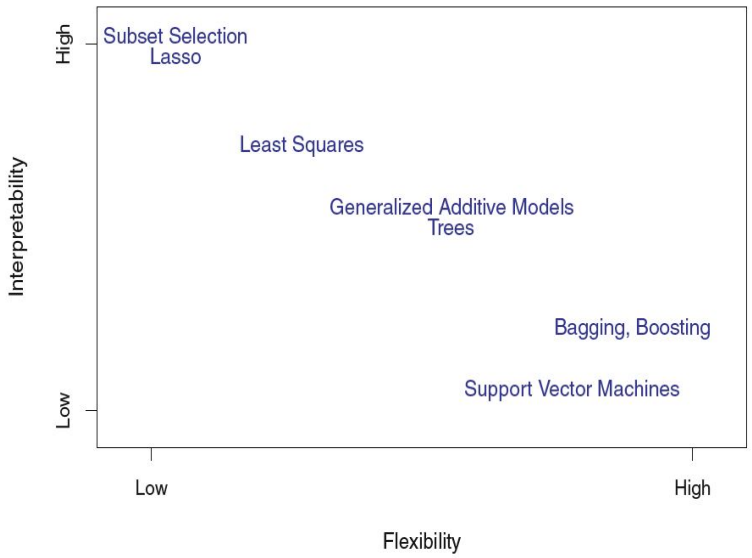
Key: DO NOT make explicit assumptions about the functional form of f, so a model can get as close to the data points as possible without being TOO wiggly

Examples: K-nearest neighbors, decision trees, support vector machines

Advantages: Can fit many functional shapes,can result in more accurate prediction performance

Disadvantages: More computationally expensive methods, less interpretability of results

Prediction Accuracy vs Interpretability



"For every 1% increase in the city's proportion of young people, sales will increase by $5000 on average" - Try doing this for Support Vector Machines or other Low Interpretability models

Because of this, lower flexibility models (like regression) are used for **inference** to learn more in-depth about predictors while higher flexibility models are often used when one solely cares about **prediction**

Supervised vs Unsupervised Learning

Supervised Learning: Use a data set X to **predict** or **detect association** with a response y
- Regression
- Classification

Unsupervised Learning: Discover signals/patterns in X, or detect associations within X
- Dimension Reduction
- Clustering

Regression vs. Classification

Regression predicts the value of a **continuous variables** based on values of predictor variables, assuming a linear or nonlinear model of dependency
- Predicting sales, wind velocities, stock market indices

Classification predicts the value of a **categorical variable** as a function as the value of other variables
- Predict whether someone will have a heart attack
- Predict if an email is spam or not

There exists a gray area - logistic regression vs. binary classification

Logistic **regression** produces a continous response between 0 and 1, but the value is most often used as a probability in binary **classification** (most commonly above 0.5 being true and below 0.5 being false)

Assessing Model Accuracy

Key: There is no model that performs better on every dataset, so we often test multiple models and find one that maximizes quality of fit

Most commonly done through **Mean Squared Error** (MSE):

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}\,(x_i))^2$$

Distinction between *training* and *test* MSE - it is most important to find model that **minimizes the *test* MSE** (e.g for new data points) For ALL models, the best way to assess performance is to check the test MSE

Assessing Model Accuracy cont.

What if we don't have test data to use? Can we just minimize training MSE to find the best model?

NO because of **overfitting**: When a model is TOO flexible, it fits TOO closely to training data points by detecting patterns likely caused by random chance, so it will hardly have any training error but will predict new data poorly

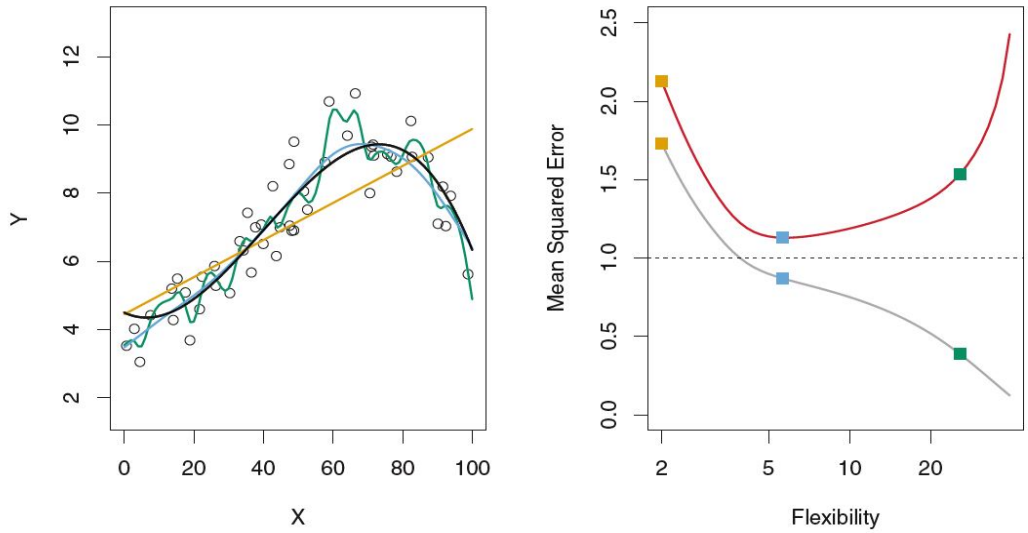


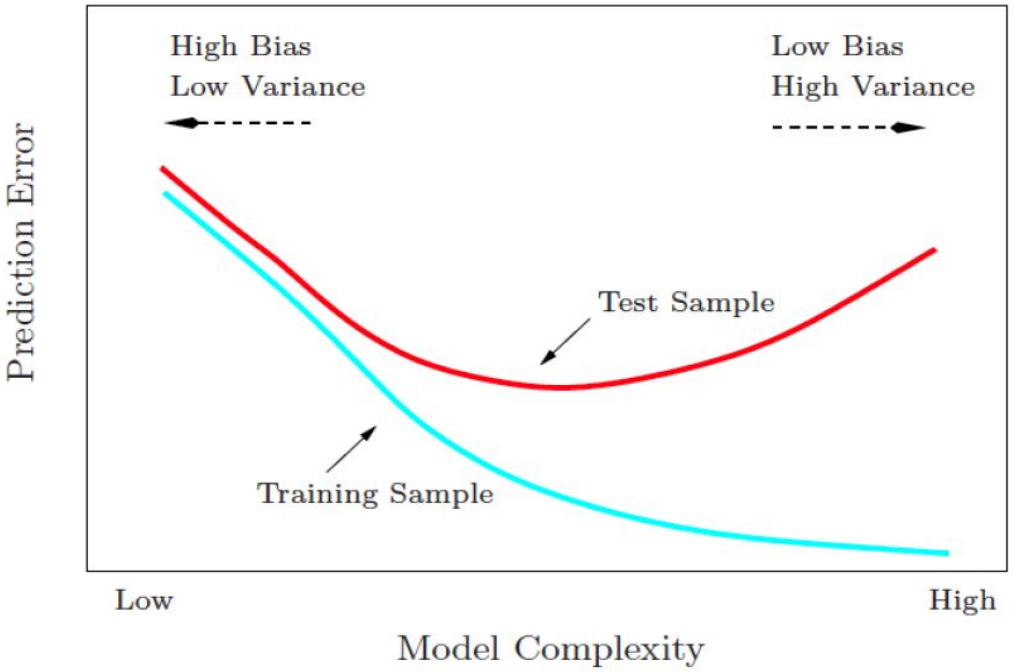
FIGURE 2.9. Left: *Data simulated from f, shown in black. Three estimates of f are shown: the linear regression line (orange curve), and two smoothing spline fits (blue and green curves).* Right: *Training MSE (grey curve), test MSE (red curve), and minimum possible test MSE over all methods (dashed line). Squares represent the training and test MSEs for the three fits shown in the left-hand panel.*

Bias-Variance Tradeoff

Expected Test MSE = Variance + Bias^2

Variance: Amount $\hat{f}$ would change using different training sets - Higher in more flexible models

Bias: Error introduced by approximating a complicated real life problem by a simpler model like linear regression -Higher in less flexible models



The Classification Setting

A **Bayes Classifier** is a very simple classifier that assigns a test observation with predictor vector x_0 to the class j for which

$$Pr(Y = j|X = x_0)$$

Error is defined simply by the proportion of points that are misclassified

In a setting with two classes,

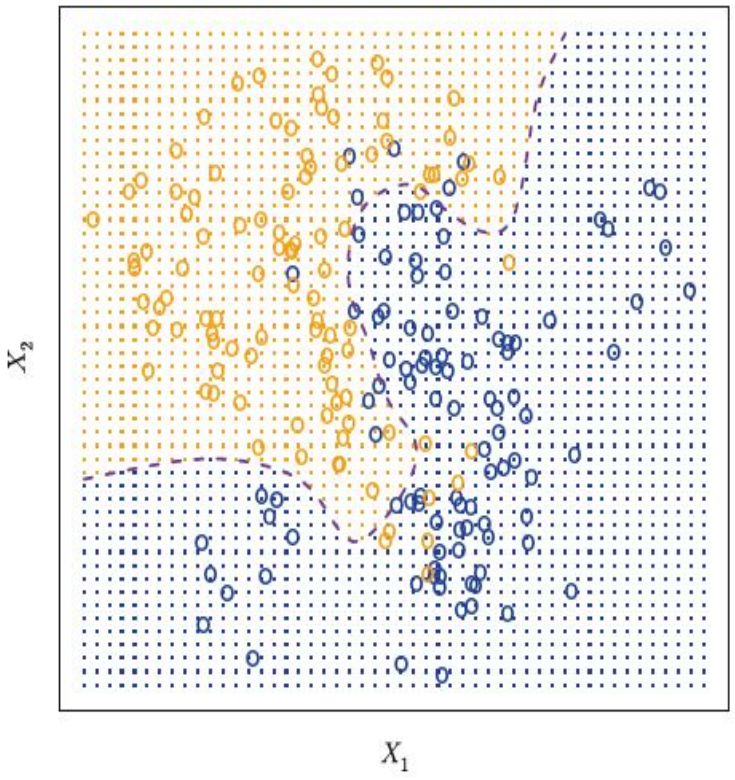
$$Pr(Y = 1|X = x_0) > 0.5$$

determines the class a new point will be assigned to.

Due to this, **classfication error** is simply defined as the proportion of points that are misclassfied by the model

The Classification Setting

A **Bayes Decision Boundary** is the "50% probability line" that separates observations predicted in two classes



This picture does not correspond to any specific model, but rather shows generally how points could be classified in a low-dimensional space

K-nearest Neighbors

A simple, common **nonparametric** classification method that implements the Bayes classifier

Use the K nearest points of the observation to predict its y-value:

$$P(Y = j|X = x_0) = \frac{1}{n} \sum_{i \in N_0} I(y_i = j)$$

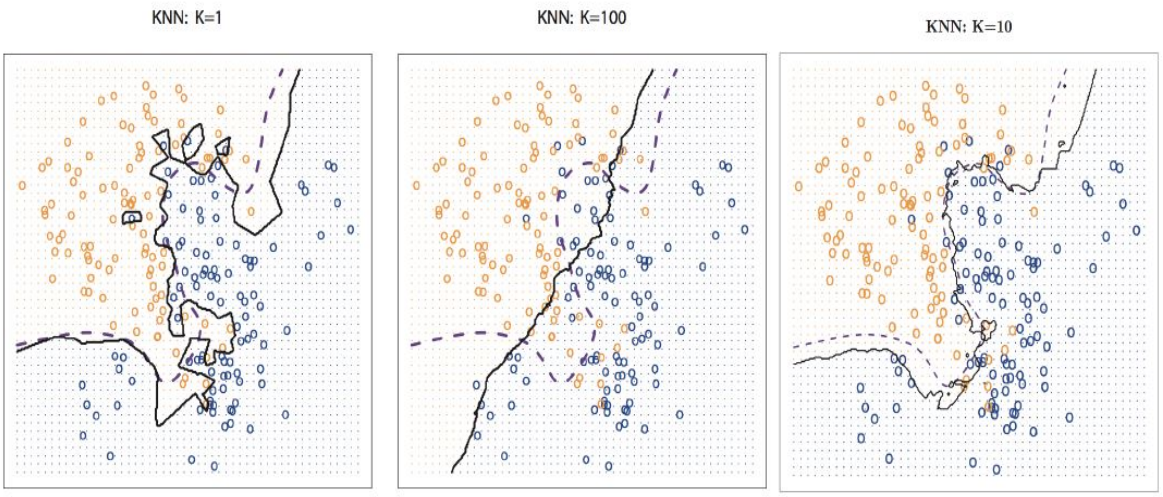
where N_0 is the neighborhood (nearest points)

Small K: Very flexible classifier, low bias and high variance

Large K: Less flexible classifier, smoother decision boundary, high bias, low variance

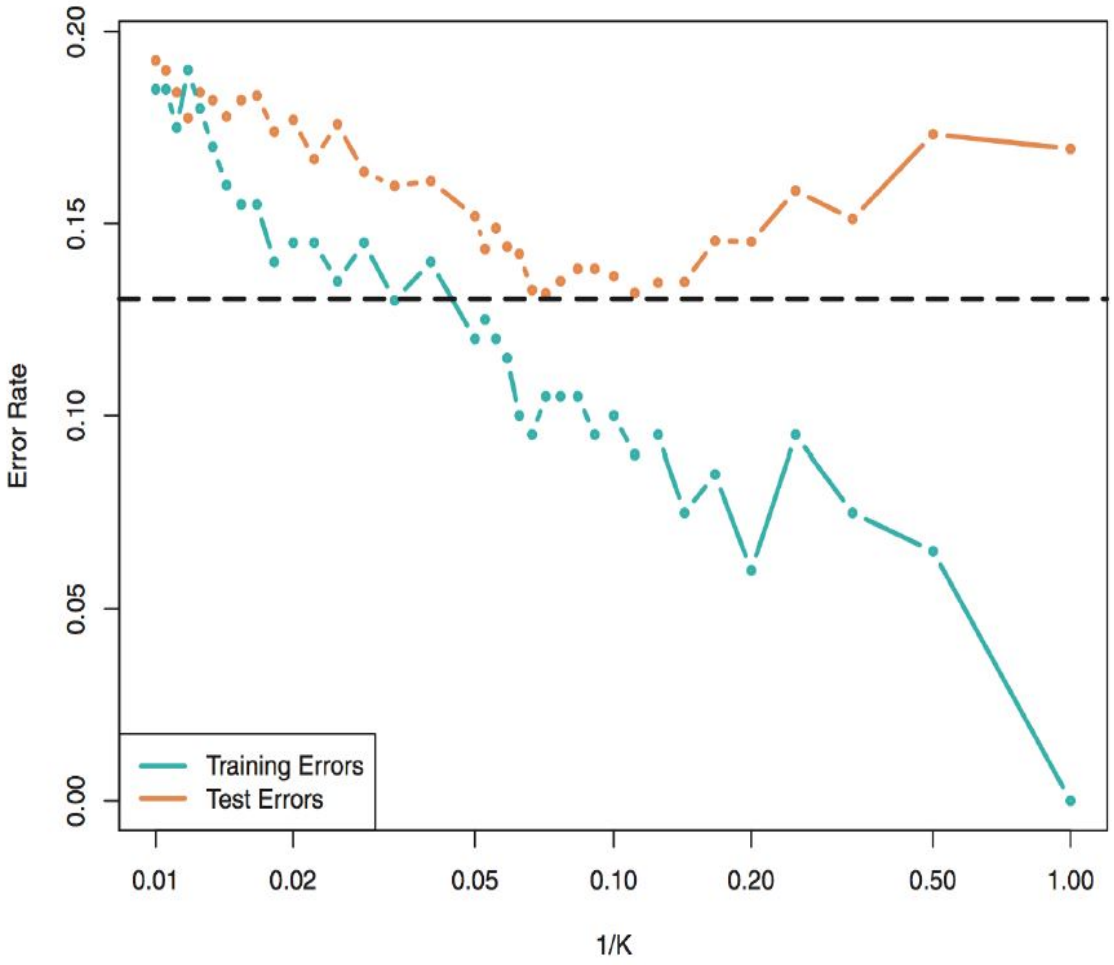
KNN Bias-Variance Tradeoff

Goal: Choose K that gives **lowest test error**



Here, the dotted line is the **optimal Bayes classifier** and the solid line is the boundary produced from the KNN model. It appears that K = 10 provides the best balance in flexibility and could best predict new observations

KNN Training vs Test Errors



Note: X axis is labeled as $1/K$, meaning X increases as K decreases.

Recommended Problems

2.1, 2.2, 2.3, 2.7

Chapter 3: Linear Regression

Ishan Shah

October 20, 2018

Linear Regression

Sometimes dull compared to modern statistical methods, but is the foundation for many of those methods and is important to understand thoroughly for both prediction and inference.

Example:

```
##      TV radio newspaper sales
## 1 230.1  37.8      69.2  22.1
## 2  44.5  39.3      45.1  10.4
## 3  17.2  45.9      69.3   9.3
## 4 151.5  41.3      58.5  18.5
## 5 180.8  10.8      58.4  12.9
## 6   8.7  48.9      75.0   7.2
```

In this `Advertising` dataset, each row represents one product . TV, `radio`, and `newspaper` represent advertising budgets (in thousands) for each medium, and `sales` are the sales for the product.

How strong is the relationship between budget and sales? How accurately can we estimate the effect of each medium on sales? Is there synergy among the advertising media? *Are the relationships linear?*

Simple Linear Regression

Predicting Y based on single predictor X

$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$

$\hat{y}$ is a prediction of Y on the basis of $X = x$

How do we find the $\hat{\beta}$ coefficients? **Least Squares approach**

Goal: Minimize RSS (Residual sum of squares)

RSS = $e_1^2 + e_2^2 + \ldots + e_n^2$

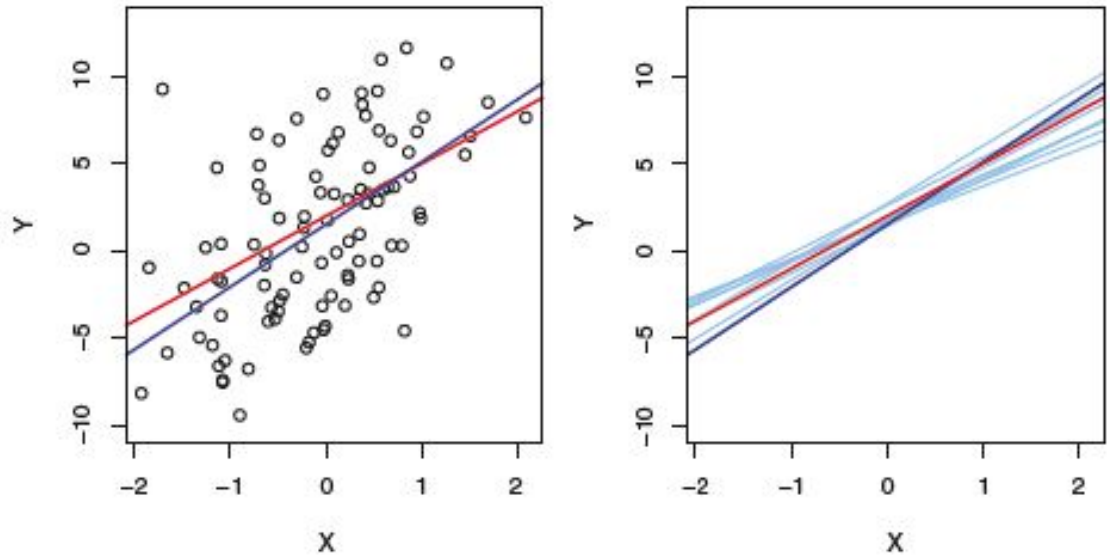
After some calculus … Least Square coefficients for simple linear regression:

$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$

$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$

Where $\bar{x}$ and $\bar{y}$ are the sample means

Unbiased Estimator



On the left image, the red line is the true population regression line, while the blue line is the least squares regression line. The right image shows many simulated least square regression lines from samples of the population. Each line is different, but on average, the lines are close to the population regression line.

This indicates that the coefficients $\hat{\beta}_1$ and $\hat{\beta}_0$ are **unbiased estimators** of true coefficients β_1 and β_0

Multiple Linear Regression

Significantly more often, there will be multiple (p) distinct predictors for response variable Y

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon$$

β_j where j represents the jth predictor is the average effect on Y with a one unit increase in X_j, *holding all other predictors fixed*

Similar to Simple Linear Regression, we find $\beta_1, \ldots \beta_p$ such that you minimize:

$$\sum_{i=1}^{n} (y_i - (\beta_1 X_{\text{i1}} + \beta_2 X_{\text{i2}} + \ldots + \beta_p X_{\text{ip}}))^2$$

This is just the **RSS** in a multi-linear setting.

Inference of Model Parameters

Do the coefficients generalize well to the population? We can use a *t-test*

$$t = \frac{\hat{\beta}}{SE(\hat{\beta})}$$

to test whether

$H_0 : \beta = 0$ vs $H_a : \beta \neq 0$

Generalized Test for All coefficients: F test

$H_0 : \beta_1 = \beta_2 = \ldots = \beta_p = 0$ vs H_a : at least one B_j is non-zero

$F = \frac{(TSS - RSS)/p}{RSS/(n-p-1)}$ where:

$TSS = \sum(y_i - \bar{y})^2$ and $RSS = \sum(y_i - \hat{y_i})^2$

All these methods are standard mainly when $p << n$ (low-dimensional settings)

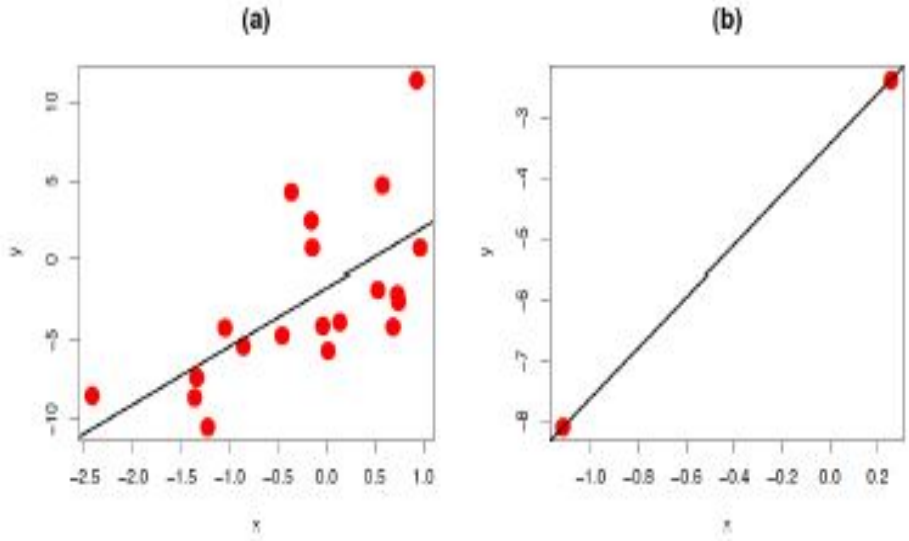
If p-value from these two tests is below desired α (often 0.05 or 0.01), we can reject the H_0

The Problem in Higher Dimensions

Common measures of error, **training MSE** and R^2, **will always improve as more variables are added into the model**

In other words - as the number of features increase, the R^2 will always increase and training MSE will always decrease.

Infact, when $n \leq p$, a **linear regression model will fit the training data perfectly** - a simple example below



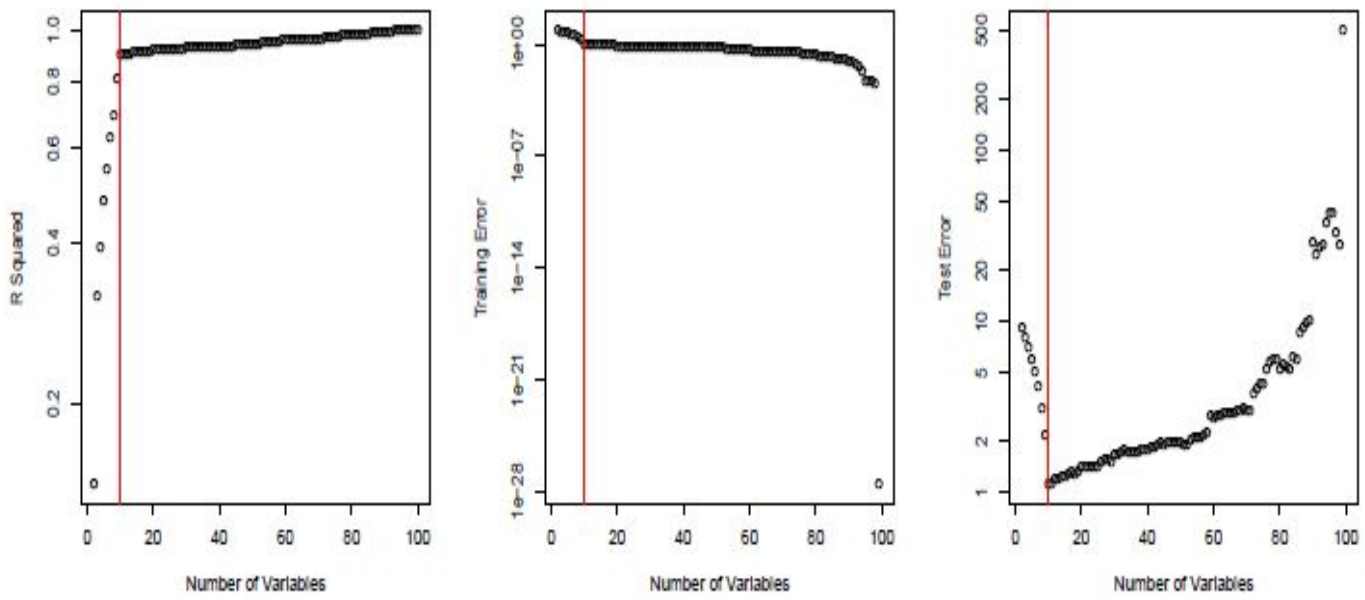
What Does This Mean

We need to use **test data** to test model performance:

Minimizing $(y_{\text{test}} - \hat{y}_{\text{test}})^2$ where

$\hat{y}_{\text{test}} = \hat{\beta}_1 X_{\text{test},1} + \ldots + \hat{\beta}_p X_{\text{test,p}}$

The average of $(y_{\text{test}} - \hat{y}_{\text{test}})^2$ over many observations is the **test error**



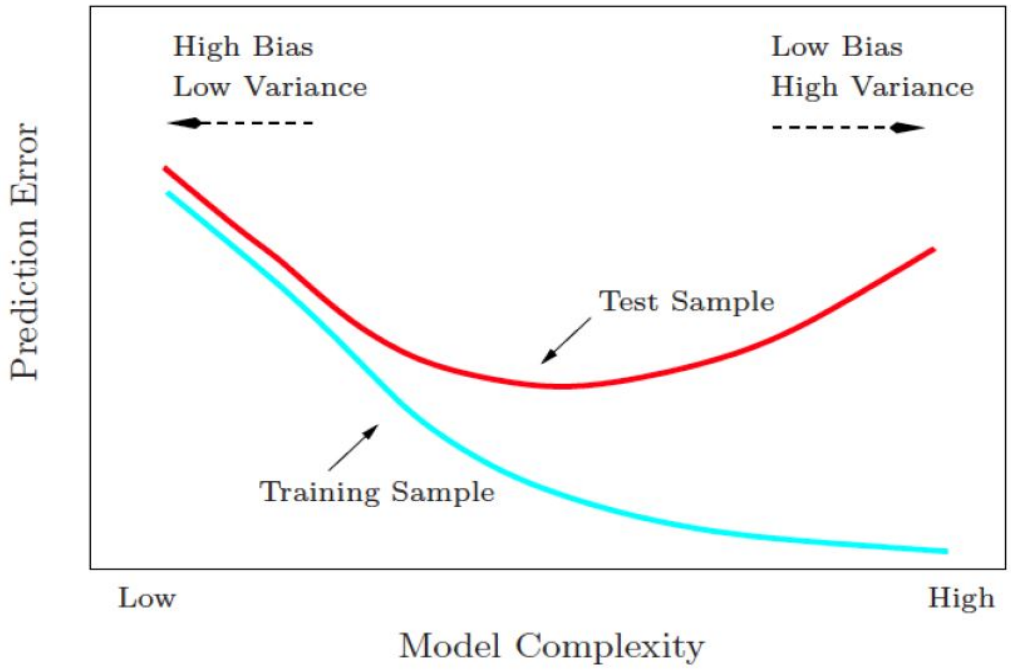
This image shows many simulated linear models of varying number of variables over random training and test datasets. You can see R^2 increase and training MSE decrease to nearly zero. However, the test error, starts to increase at a certain point again. *Note the logarithmic scales*

Bias-Variance Tradeoff

As complexity increases:

Bias of $\hat{\beta}$ will decrease and variance of $\hat{\beta}$ will increase

Reminder:



^Compare this to the graph on the previous slide

Methods to Select Coefficients

Cannot test every practical model! There are 2^p models so we need an automated and efficient approach:

Forward Selection:
1. Test p linear regressions (one for each variable)
2. Add in coefficient with lowest RSS to model
3. Test every 2-variable model with first variable in the model, pick one with lowest RSS
4. Repeat until ending condition

Backward Selection::
1. Begin with model with all variables in
2. Remove variable with largest p-value
3. Fit $p - 1$ variable model, remove next highest p-value
4. Repeat until ending condition

Ridge and Lasso Regression: Shrinks coefficients, more on this later

Recommended Problems

3.3, 3.4, 3.8, 3.9, 3.10, 3.14

Chapter 5: Resampling Methods

Ishan Shah

November 12, 2018

Overview

Resampling Methods involve drawing multiple samples from a training set and refitting a model of interest on each sample to obtain additional information about the model.

Takes two forms: **Cross-Validation** and **Bootstrap** - we will primarily focus on **Cross-Validation**

Main uses:
Model Assessment: Evaluates a model's performance with a more accurate test error measure
Model Selection: Selecting the appropriate level of flexibility for a model

Cross Validation

Premise: We rarely have a large, given test set we can use to test the accuracy of our model. We need a way to assess and select our model from just a training set.

Three Approaches:
1. Validation Set
2. Leave-one-out cross-validation
3. K-Fold cross-validation

We have to split our dataset into a training and test set and lock the test set away until we finalize a model on the training set.

Validation Set

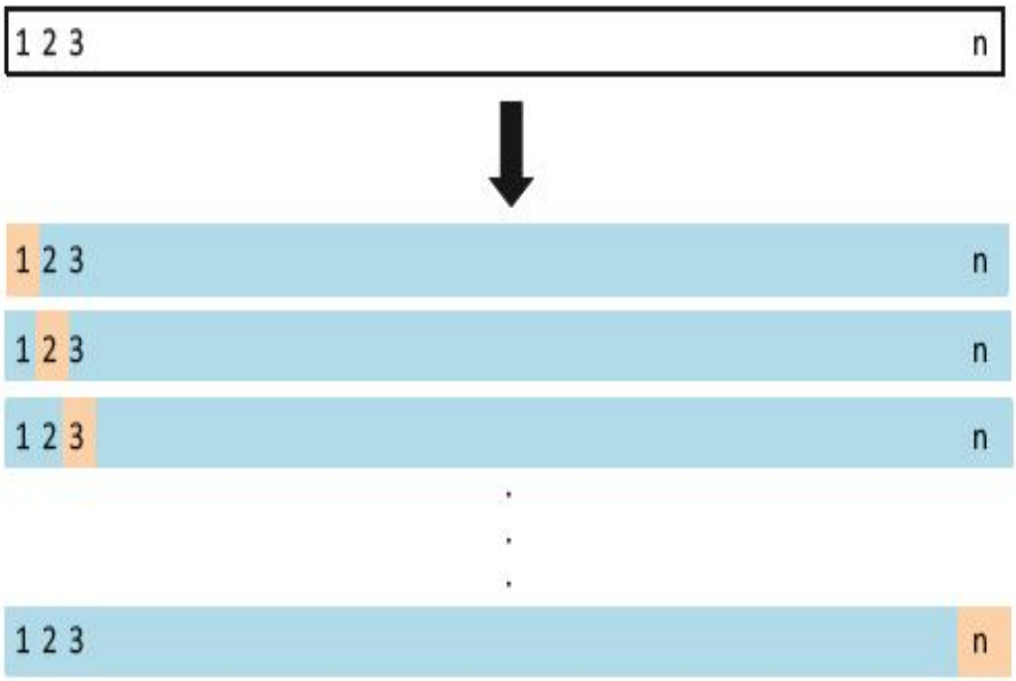
1. Randomly split the data 50/50 into a **training set** and a **validation set**
2. Train model on training set and run on validation set to obtain a **test MSE**
3. Repeat multiple times if desired

Advantage: Simple to implement and computationally inexpensive

Disadvantages: Test error rate can be highly variable with multiple repetitions, as it depends on precisely how the observations are split

Only a subset of observations are used to train model, leading to an *overestimate* of test error

LOOCV (Leave-One-Out Cross Validation)



LOOCV Continued

Run model on $n-1$ observations (test set) and predict the outcome of the one remaining point.

Find MSE of that one point, and then repeat for the rest of the observations.

Overall LOOCV test MSE estimate is the average of n test error estimates:

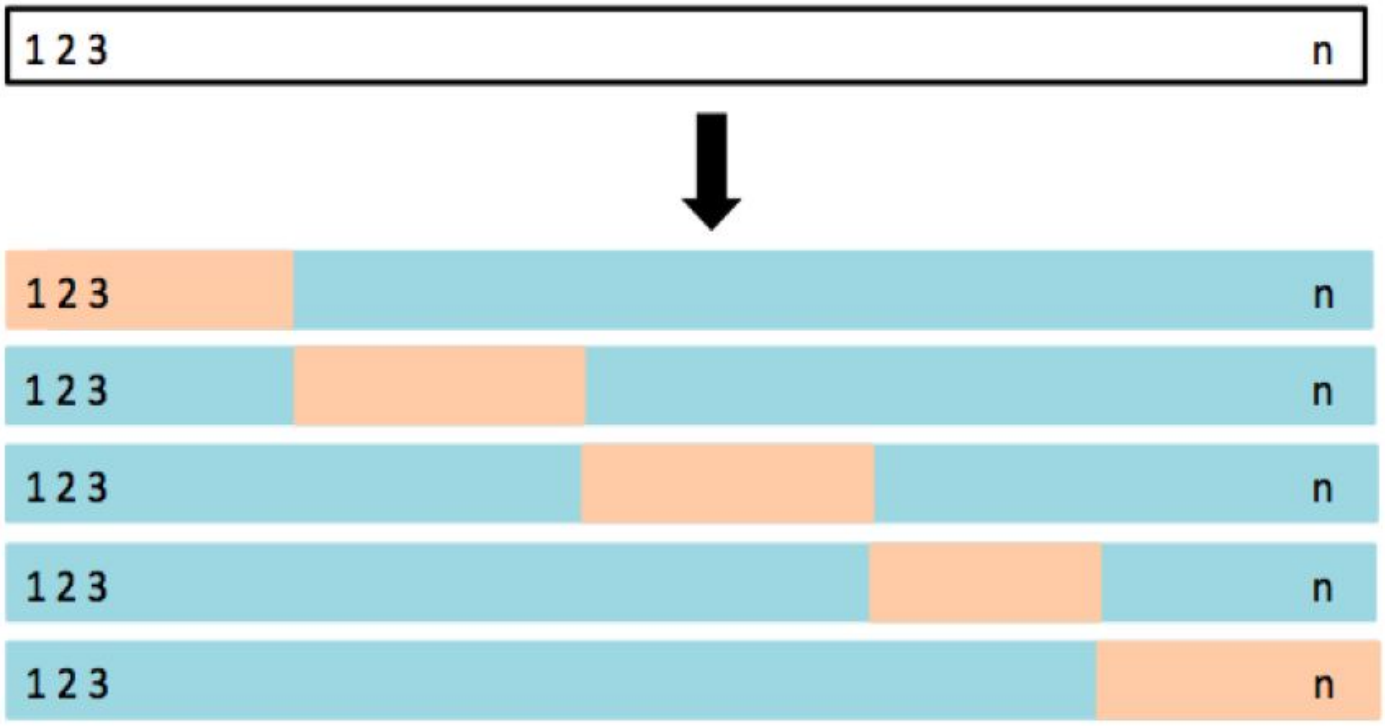
$$CV_{(\mathrm{n})} = \frac{1}{n}\sum_{i=1}^{n} MSE_i$$

Advantages: Uses all the data to train models, preventing test error overestimation
Little variability in final test error result

Disadvantages: *Except linear regression*, can be very computationally expensive, especially for complex models

K-Fold Cross Validation

An example of **5-fold** Cross Validation



K-Fold CV Continued

1. Divide data into k groups of equal sizes
2. Treat the first group as the validation set and the other $k-1$ groups combined as the training set
3. Find Test MSE
4. Repeat with each group being the test set once
5. Average Test MSE's as shown:

$$CV_{(\mathrm{k})} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

Often the preferred method, especially for complex models, essentially a compromise between LOOCV and validation-set approach; not too computationally expensive but still uses all the data to train the model.

Recommended Problems

5.8, 5.9

CH5 Example 8

Ishan Shah

December 4, 2018

Generate a simulated data set as follows:

```
set.seed(1)
x = rnorm(100)
y = x-2*x^2+rnorm(100)
```

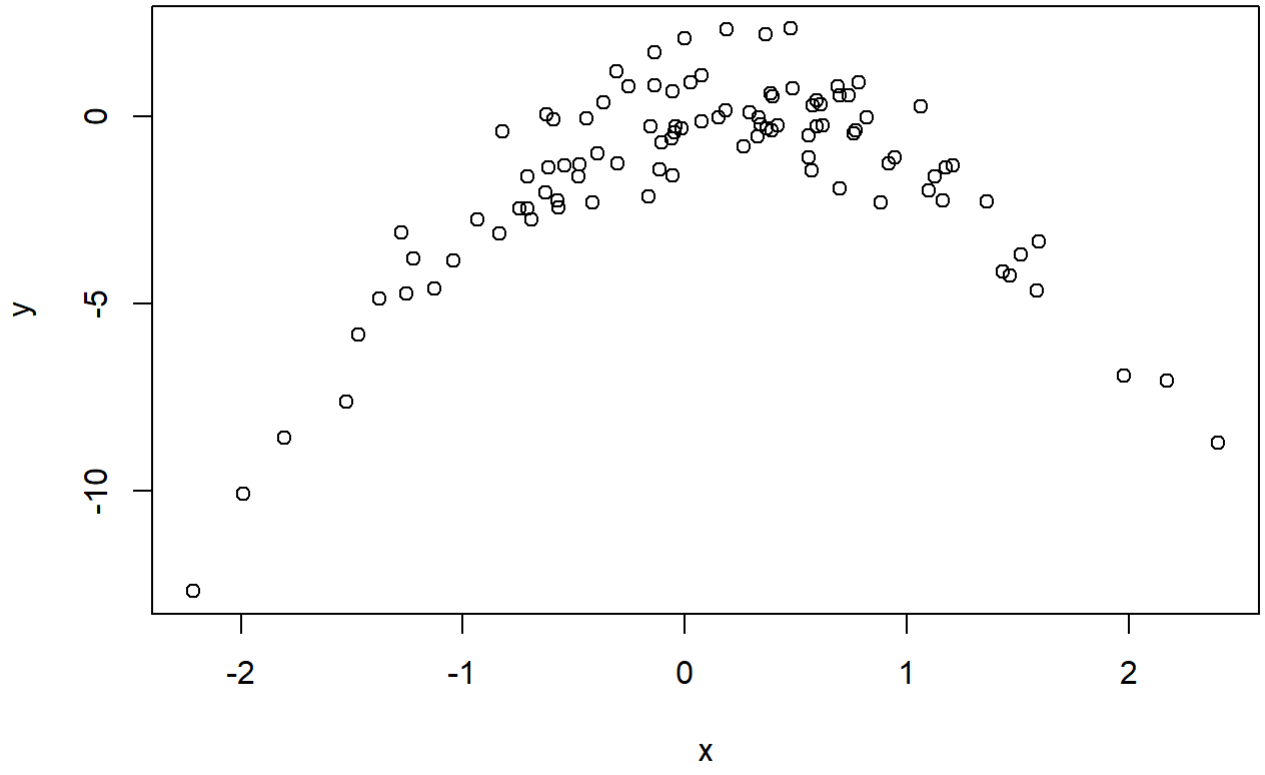
(a) In this data set, what is n and what is p? Write out the model used to generate the data in equation form.

Here, $n = 100$ and $p = 2$, the equation is:

$Y + X - 2X^2 + \epsilon$

(b) Create a scatterplot of X against Y. Comment what you find.

```
plot(x,y)
```



There is a clear quadratic relationship between x and y, with the variation occurring from the error terms that originated from the `rnorm` function.

(c) Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares:

$Y = \beta_0 + \beta_1 X + \epsilon$

```
set.seed(12345)
data <- data.frame(x,y)
lm1 <- glm(y ~ poly(x,1))
cv.glm(data,lm1)$delta[1]
```

```
## [1] 7.288162
```

$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$

```
lm2 <- glm(y ~ poly(x,2))
cv.glm(data,lm2)$delta[1]
```

```
## [1] 0.9374236
```

$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$

```
lm3 <- glm(y ~ poly(x,3))
cv.glm(data,lm3)$delta[1]
```

```
## [1] 0.9566218
```

$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$

```
lm4 <- glm(y ~ poly(x,4))
cv.glm(data,lm4)$delta[1]
```

```
## [1] 0.9539049
```

(d) Repeat (c) using another random seed, and report your results. Are your results the same as what you got in (c) ? Why ?

$Y = \beta_0 + \beta_1 X + \epsilon$

```
set.seed(54321)
data <- data.frame(x,y)
lm1 <- glm(y ~ poly(x,1))
cv.glm(data,lm1)$delta[1]
```

```
## [1] 7.288162
```

$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$

```
lm2 <- glm(y ~ poly(x,2))
cv.glm(data,lm2)$delta[1]
```

```
## [1] 0.9374236
```

$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$

```
lm3 <- glm(y ~ poly(x,3))
cv.glm(data,lm3)$delta[1]
```

```
## [1] 0.9566218
```

$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$

```
lm4 <- glm(y ~ poly(x,4))
cv.glm(data,lm4)$delta[1]
```

```
## [1] 0.9539049
```

The errors are the exact same for each of the models despite the different random seed. This is because LOOCV evaluates models excluding each of the n observations once. Therefore, there is no aspect of randomness and a seed won't change anything.

(e) Which of the models in (c) had the smallest LOOCV error ? Is this what you expected ? Explain your answer.

The LOOCV Error is lowest in the second (quadratic) model. This is what is expected because the original equation is quadratic and the scatterplot clearly shows a quadaratic relationship.

(f) Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (c) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results?

To see the significance of all the coefficients, we will show a summary of the 4th degree model.

```
summary(lm(y ~ poly(x,4), data = data))
```

```
##
## Call:
## lm(formula = y ~ poly(x, 4), data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.0550 -0.6212 -0.1567  0.5952  2.2267
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
## poly(x, 4)1   6.18883    0.95905   6.453 4.59e-09 ***
## poly(x, 4)2 -23.94830    0.95905 -24.971  < 2e-16 ***
## poly(x, 4)3   0.26411    0.95905   0.275    0.784
## poly(x, 4)4   1.25710    0.95905   1.311    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9591 on 95 degrees of freedom
## Multiple R-squared:  0.8753, Adjusted R-squared:  0.8701
## F-statistic: 166.7 on 4 and 95 DF,  p-value: < 2.2e-16
```

Here, the 1st and second degree coefficients are clearly statistically significant ($p < 0.05$), whereas the third and fourth degree coefficients are not ($p > 0.05$). This validates the cross-validation results, as the error started to increase slightly when the 3rd and 4th degree coefficients were added.

Chapter 6: Model Selection and Regularization

Ishan Shah

January 21, 2019

Overview

The standard linear model: $Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p$

We have learned about the **least squares** method to find optimal coefficeients, but there are alternative fitting procedures that can yield better *prediction accuracy* and *model interpretability*, including

Subset selection: Finding an optimal subset of p coefficients, and then using least squares

Shrinkage: Coefficients' values can be reduced significantly to lower variance in the model - takes the form of Ridge (cannot shrink coefficients to zero) or LASSO (can shrink coefficients to zeros)

Dimension Reduction: Where, $M < p$, we project p predictors into an M-dimensional subspace, then use these M projections as predictors

Consequences on Accuracy and Interpretability

Prediction Accuracy: Given the true relationship of the model is approximately linear, where n is the number of observations and p is the number of predictors, if

$n >> p$: **Least squares** tends to have low variance and perform well on test observations.

$n > p$: Potential for lots of variability in least squares and overfitting, consider **shrinkage, subset selection**

$n <= p$: Least squares cannot function, must use **shrinkage** for model to operate and have good prediction accuracy

Model Interpretability:
Removing irrelevant variables through susbset selection or shrinkage methods often paints a clearer picture of the true relationships in the linear model and increases interpretability.

Subset Selection

Best Subset Selection: Performs least squares on every possible combination of predictors (2^p models in total) and selects the best-performing model based on certain criteria - this method is conceputually simple but **very** computationally expensive. Even modern, fast computers cannot feasibly run this with p > 40, so instead we can do…

Forward Selection:
1. Test p single-predictor linear regressions
2. Add in predictor with lowest RSS to model
3. Test every 2-predictor model with first predictor in the model, pick one with lowest RSS
4. Continue adding variables up to p predictors
5. Once there is a best model for each of $1, 2, \ldots, p$ predictors, choose overall best model based on ending criteria (next slide) or cross-validation

Backward Selection::
1. Begin with model with all variables in
2. Remove each predictor, testing every possible model with $p - 1$ predictors, move to $p - 1$ model with lowest RSS
3. Fit $p - 2$ predictor models, move to the one with lowest RSS
4. Continue dropping variables until 1 predictor
5. Once there is a best model for each of $1, 2, \ldots, p$ predictors, choose overall best model based on ending criteria (next slide) or cross-validation.

These iterate through a maximum of $1 + p(p + 1)/2$ models. To show the difference, when $p = 20$, best subset selection tests over 1 million models where forward/backward selection fit at most 211 models.

Ways to Recognize the "best" Model

Method 1: Make an *adjustment* to training error to account for overfitting bias through either **minimizing** C_p, *Akaike information criterion* (AIC), *Bayesian information criterion* (BIC), or **maximizing** adjusted R^2.

Where d is the number of predictors, RSS = Residual Sum-of-Squares ($\sum(y_i - \bar{y_i})^2$), TSS = Total sum of squares ($\sum(y_i - \bar{y})^2$), $\hat{\sigma}^2$ is estimate of variance of error ϵ, and n is number of observations,

C_p = $\frac{1}{n}(RSS + 2d\hat{\sigma}^2)$ - adds a penalty to training RSS to adjust for training error underestimating test error

AIC = $\frac{1}{n\hat{\sigma}^2}(RSS + 2d\hat{\sigma}^2)$ - proportional to C_p

BIC = $\frac{1}{n}(RSS + log(n)d\hat{\sigma}^2)$ - adds heavier penalty to models with more variables, tends to select fewer variables

Adjusted R^2 = $1 - \frac{RSS/(n-d-1)}{TSS/(n-1)}$ - addition of $n - d - 1$ term adds a cost to regular R^2 for the inclusion of more variables in the model

Ways to Recognize the "best" Model

Method 2: Use Validation set or Cross-Validation to directly estimate test error

After we have a few models in consideration, we can use these approaches from chapter five and see which produces the lowest test MSE.

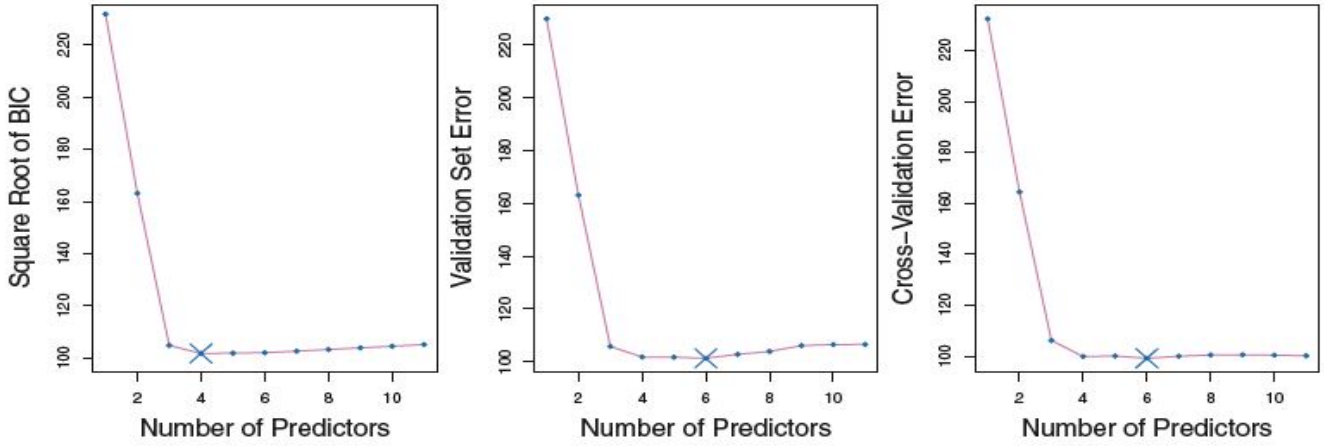


FIGURE 6.3. *For the* `Credit` *data set, three quantities are displayed for the best model containing d predictors, for d ranging from* 1 *to* 11. *The overall* best *model, based on each of these quantities, is shown as a blue cross.* Left: *Square root of BIC.* Center: *Validation set errors.* Right: *Cross-validation errors.*

This is a comparison of the models selected using BIC, validation set approach, and cross-validation. BIC clearly selects a model with fewer coefficients (4), while validation set and cross-validation select 6-variable models.

Shrinkage Methods - Ridge Regression

While least squares minimizes RSS, ridge regression finds $\hat{\beta}_\lambda^R$ values through minmizing

$$\sum_{i=1}^{n}(y_i - \beta_0 - (\beta_1 X_{\text{i1}} + \beta_2 X_{\text{i2}} + \ldots + \beta_p X_{\text{ip}}))^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

=

$$RSS + \lambda \sum_{j=1}^{p} \beta_j^2$$

,

where $\lambda \geq 0$ is a *tuning parameter*, meaning it must be tweaked to a certain value to find the optimal model. One finds the optimal λ through cross-validation guess-and-check.

When λ is zero, $\hat{\beta}_\lambda^R$ values are least-squares estimates, but as λ approaches infinity, ridge coefficients approach zero.

$||\beta||_2$ is known as the ℓ_2 norm, defined as $\sqrt{\sum_{j=1}^{p} \beta_j^2}$. As λ increases, the ℓ_2 norm decreases, and so does $||\hat{\beta}_\lambda^R||_2 / ||\hat{\beta}||_2$, which ranges from zero (coefficients shrunk to zero) to 1 (no shrinking).

Scaling of any variable **does** affect ridge coefficients, unlike least squares, meaning it is important to **standardize** variables before implementing this.

Shrinkage Methods - Ridge Regression

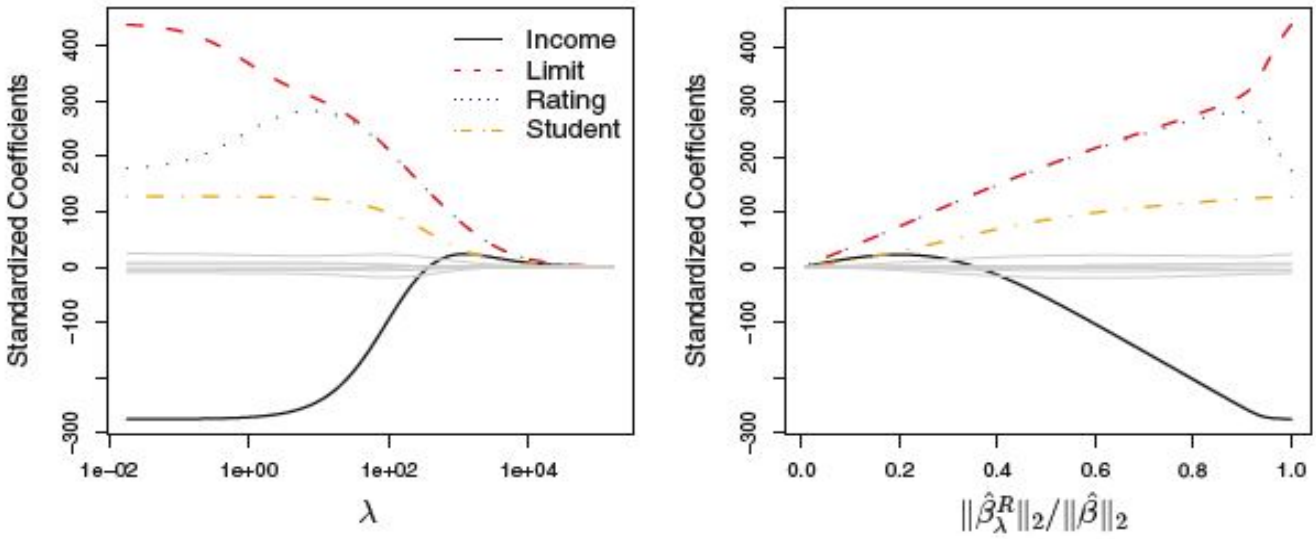


FIGURE 6.4. *The standardized ridge regression coefficients are displayed for the* `Credit` *data set, as a function of* λ *and* $\|\hat{\beta}^R_\lambda\|_2/\|\hat{\beta}\|_2$.

Clearly, as the λ value increases and $||\hat{\beta}^R_\lambda||_2/||\hat{\beta}||_2$ decreases, the coefficients are shrunk almost all the way to zero. The optimal point for these coefficients are somewhere in the middle of this graph.

Ridge Regression Bias-Variance Tradeoff

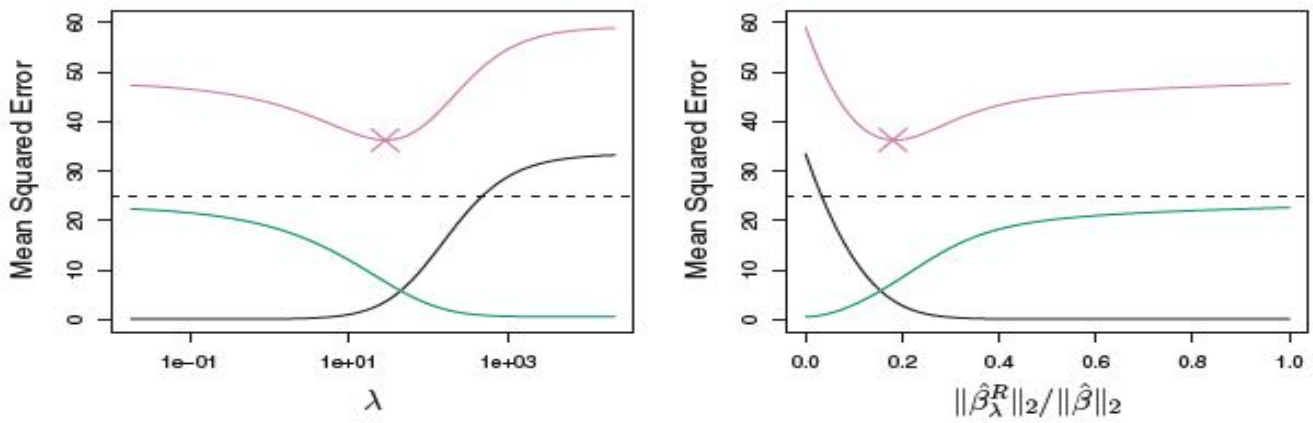


FIGURE 6.5. *Squared bias (black), variance (green), and test mean squared error (purple) for the ridge regression predictions on a simulated data set, as a function of* λ *and* $\|\hat{\beta}^R_\lambda\|_2/\|\hat{\beta}\|_2$. *The horizontal dashed lines indicate the minimum possible MSE. The purple crosses indicate the ridge regression models for which the MSE is smallest.*

The optimal λ value here is approximately 30, where both bias and variance are minimized. Clearly, the variance is significantly lower at the ridge estimate than the least-squares estimate (when $\lambda = 0$), with a cost of only a slight increase in bias. This, along with it being signifcantly less computationally expensive than best subset selection, shows the potential advantages of ridge regression, especially when the least squares estimates have high variance.

Shrinkage Methods - LASSO

The main disadvantage of ridge regression is that it cannot completely get rid of coefficients. LASSO is an alternative approach that allows for **feature selection** and the creation of sparse models.

LASSO finds $\hat{\beta}_{\lambda}^{L}$ values through minmizing

$$\sum_{i=1}^{n}(y_i - \beta_0 - (\beta_1 X_{\mathrm{i1}} + \beta_2 X_{\mathrm{i2}} + \ldots + \beta_p X_{\mathrm{ip}}))^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

=

$$RSS + \lambda \sum_{j=1}^{p} |\beta_j|$$

,

The only difference here is the $|\beta_j|$ vs the β_j^2, where $||\beta||_1 = \sum |\beta_j|$ is the ℓ_1 norm

LASSO

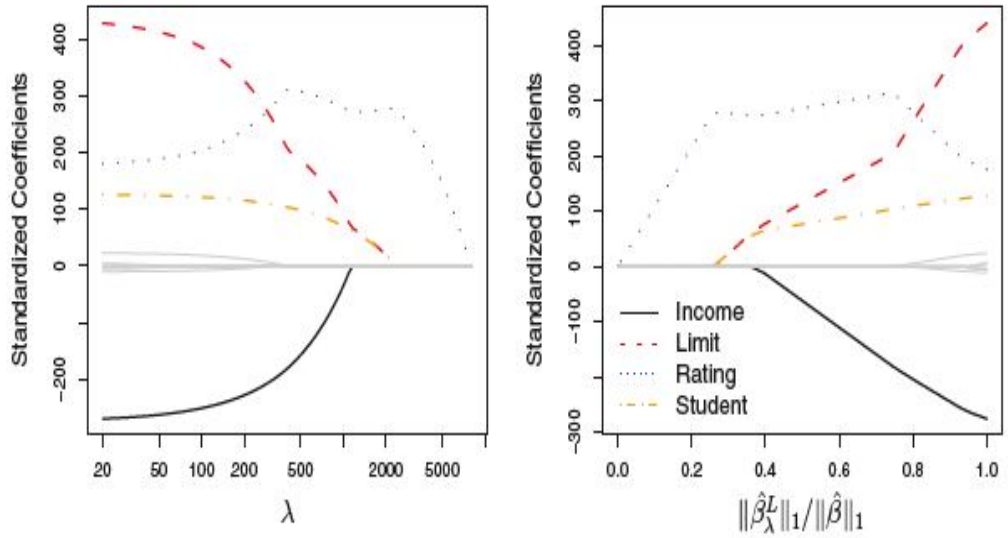


FIGURE 6.6. *The standardized lasso coefficients on the* `Credit` *data set are shown as a function of* λ *and* $\|\hat{\beta}^L_\lambda\|_1/\|\hat{\beta}\|_1$.

This graph is very similar to ridge, but the main difference being some of the coefficients are shrunk completely to zero.

Alternate Formulation for Ridge and LASSO

Minimize β

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})$$

subject to…
For LASSO:

$$\sum_{j=1}^{p}|\beta_j| \leq s$$

For Ridge:

$$\sum_{j=1}^{p}\beta_j^2 \leq s$$

s has a inverse effect of λ. Think of it as a budget. If it is small, the $\hat{\beta}$ values have to shrink to fit within that budget.

Ridge vs. LASSO

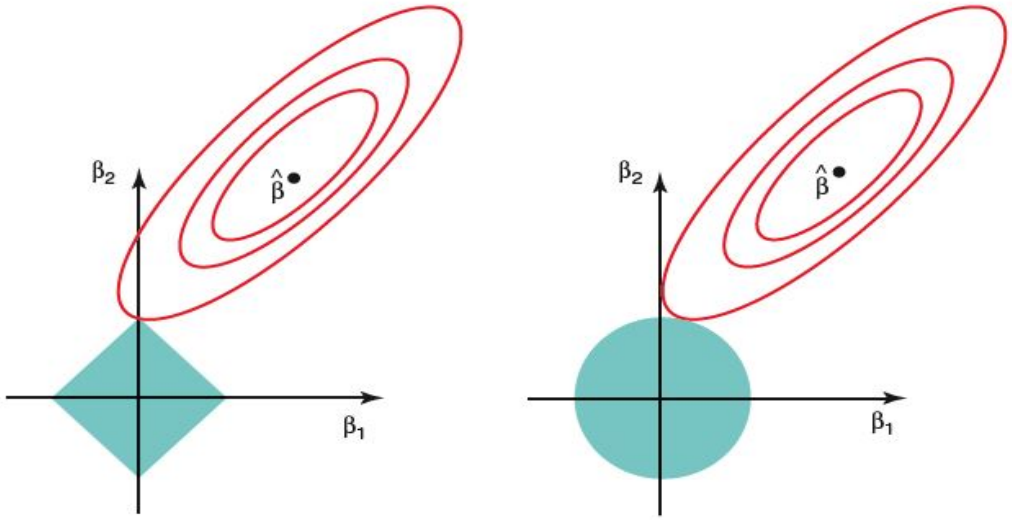


FIGURE 6.7. *Contours of the error and constraint functions for the lasso* (left) *and ridge regression* (right). *The solid blue areas are the constraint regions,* $|\beta_1| + |\beta_2| \leq s$ *and* $\beta_1^2 + \beta_2^2 \leq s$, *while the red ellipses are the contours of the RSS.*

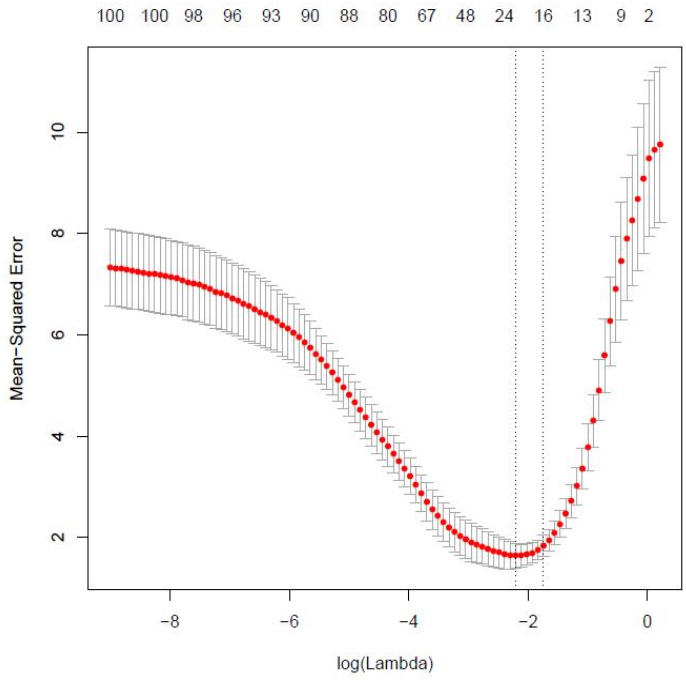
This geometric representation of ridge vs. LASSO shows how coefficients can be shrunk to exactly zero on the axes at LASSO but not ridge.

No approach is necessarily better than the other. It depends on whether all the coefficents are related to the response or not. To truly find out for your specific data, you would need to try cross-validation with both models.

Choosing lambda

To choose λ for both Ridge and LASSO, set up a grid of values usually on a log scale (ex: 0.001, 0.01, 0.1,…), and run cross validation with each value. Choose the λ with the lowest CV MSE.

Example output to choose the optimal λ -



The optimal λ in this case would be around $log(-2)$ or 0.01.

Recommended Problems

6.2, 6.3, 6.4

ISLR Ridge and Lasso Lab

Ishan Shah

February 3, 2019

Setup

We will be using the "Hitters" dataset from the ISLR package. For more info, enter `?Hitters` into the console. We should define our x here using the `model.matrix` function, which converts categorical variables in to dummy 0,1 binary variables.

```
Hitters = na.omit(Hitters) #We remove rows with NA values
x = model.matrix(Salary ~., Hitters)[,-1]
y = Hitters$Salary

summary(x)
```

```
##      AtBat            Hits           HmRun            Runs       
##  Min.   : 19.0   Min.   :  1.0   Min.   : 0.00   Min.   :  0.00  
##  1st Qu.:282.5   1st Qu.: 71.5   1st Qu.: 5.00   1st Qu.: 33.50  
##  Median :413.0   Median :103.0   Median : 9.00   Median : 52.00  
##  Mean   :403.6   Mean   :107.8   Mean   :11.62   Mean   : 54.75  
##  3rd Qu.:526.0   3rd Qu.:141.5   3rd Qu.:18.00   3rd Qu.: 73.00  
##  Max.   :687.0   Max.   :238.0   Max.   :40.00   Max.   :130.00  
##       RBI             Walks            Years            CAtBat       
##  Min.   :  0.00   Min.   :  0.00   Min.   : 1.000   Min.   :   19.0  
##  1st Qu.: 30.00   1st Qu.: 23.00   1st Qu.: 4.000   1st Qu.:  842.5  
##  Median : 47.00   Median : 37.00   Median : 6.000   Median : 1931.0  
##  Mean   : 51.49   Mean   : 41.11   Mean   : 7.312   Mean   : 2657.5  
##  3rd Qu.: 71.00   3rd Qu.: 57.00   3rd Qu.:10.000   3rd Qu.: 3890.5  
##  Max.   :121.00   Max.   :105.00   Max.   :24.000   Max.   :14053.0  
##      CHits            CHmRun           CRuns             CRBI        
##  Min.   :   4.0   Min.   :  0.00   Min.   :   2.0   Min.   :   3.0  
##  1st Qu.: 212.0   1st Qu.: 15.00   1st Qu.: 105.5   1st Qu.:  95.0  
##  Median : 516.0   Median : 40.00   Median : 250.0   Median : 230.0  
##  Mean   : 722.2   Mean   : 69.24   Mean   : 361.2   Mean   : 330.4  
##  3rd Qu.:1054.0   3rd Qu.: 92.50   3rd Qu.: 497.5   3rd Qu.: 424.5  
##  Max.   :4256.0   Max.   :548.00   Max.   :2165.0   Max.   :1659.0  
##      CWalks          LeagueN          DivisionW         PutOuts      
##  Min.   :   1.0   Min.   :0.0000   Min.   :0.0000   Min.   :   0.0  
##  1st Qu.:  71.0   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 113.5  
##  Median : 174.0   Median :0.0000   Median :1.0000   Median : 224.0  
##  Mean   : 260.3   Mean   :0.4715   Mean   :0.5095   Mean   : 290.7  
##  3rd Qu.: 328.5   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.: 322.5  
##  Max.   :1566.0   Max.   :1.0000   Max.   :1.0000   Max.   :1377.0  
##     Assists          Errors          NewLeagueN    
##  Min.   :  0.0   Min.   : 0.000   Min.   :0.0000  
##  1st Qu.:  8.0   1st Qu.: 3.000   1st Qu.:0.0000  
##  Median : 45.0   Median : 7.000   Median :0.0000  
##  Mean   :118.8   Mean   : 8.593   Mean   :0.4639  
##  3rd Qu.:192.0   3rd Qu.:13.000   3rd Qu.:1.0000  
##  Max.   :492.0   Max.   :32.000   Max.   :1.0000
```

As you can see from the last variable, the `NewLeague` categorical variable was converted to a binary variable, with 46.39% of the data points being

“N”.

Ridge Regression

In ridge regression, we must first define a vector of *tuning parameter* λ using the `seq` function. We will generate 50 evenly-spaced values on a logarithmic scale between 10^-3 and 10^6. We then use the `glmnet` function to generate ridge models for all lambdas (the function automatically standardizes the dataset's numerical values).

```
grid = 10^seq(-3, 6, length = 50) #50 evenly spaced exponents from -3 to 6
ridge.mod =glmnet (x,y, alpha =0, lambda=grid) #Creates a ridge regression model
```

Effect of Various Lambda Values

We will now see what the coefficients look like at small, medium, and large λ values.

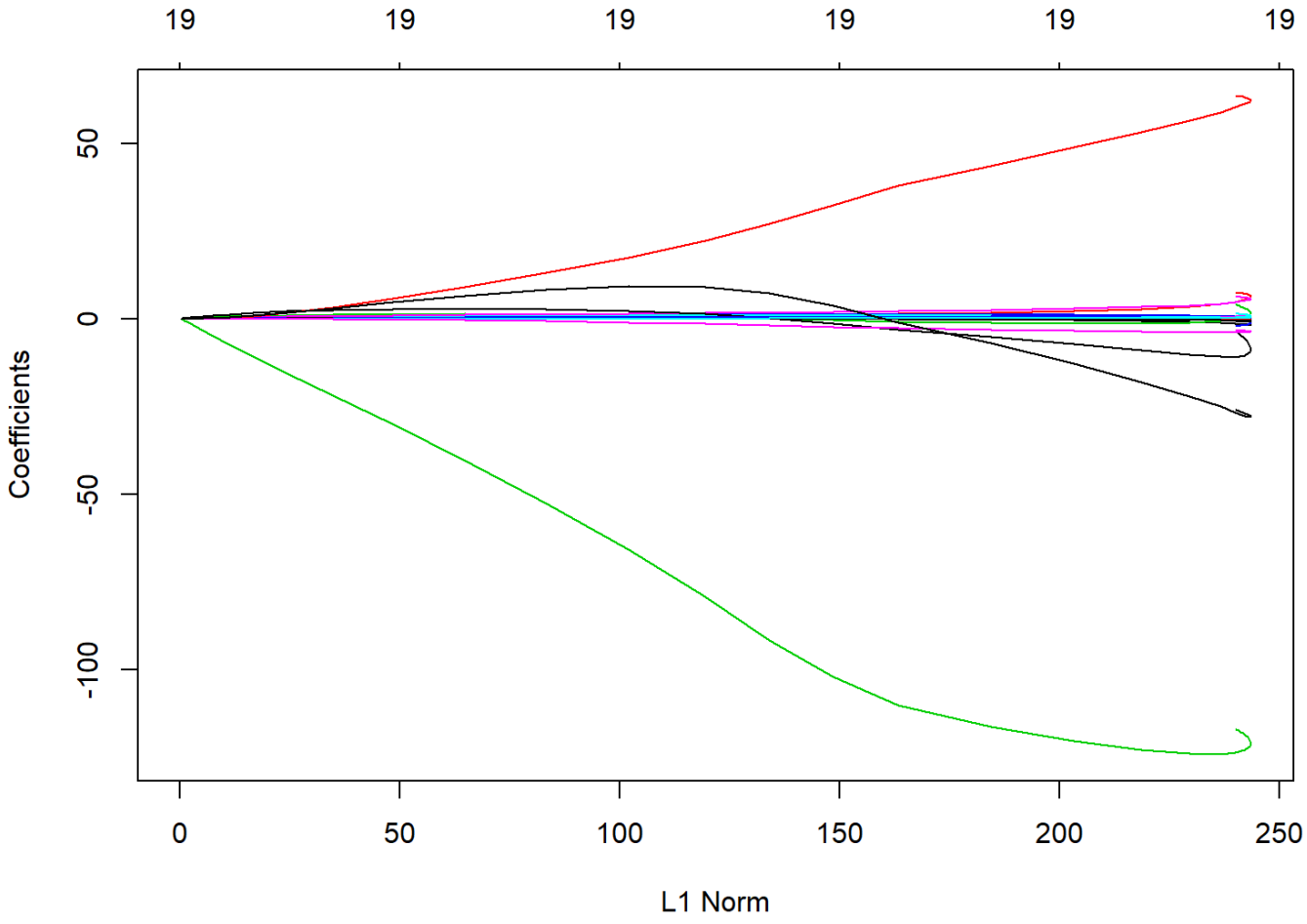
```
Lambda_Value = ridge.mod$lambda[c(1,25,50)] #Takes first, middle, and last lambdas
coeffs = rbind(coef(ridge.mod)[,1], coef(ridge.mod)[,25], coef(ridge.mod)[,50])
round(cbind(Lambda_Value, coeffs), 5) #Organizes coefficients into a table
```

```
##      Lambda_Value (Intercept)    AtBat    Hits    HmRun     Runs      RBI
## [1,]  1.00000e+06   534.04267  0.00054 0.00197  0.00795  0.00333  0.00352
## [2,]  3.90694e+01    59.72178 -0.46158 2.22342 -1.36928  1.11391  0.77824
## [3,]  1.00000e-03   164.06873 -1.97316 7.37715  3.94799 -2.20387 -0.91994
##        Walks    Years   CAtBat   CHits  CHmRun   CRuns    CRBI   CWalks
## [1,] 0.00414  0.01697  0.00005 0.00017 0.00129 0.00034 0.00036  0.00038
## [2,] 2.92953 -7.27798  0.00319 0.11525 0.64572 0.24693 0.23581 -0.19248
## [3,] 6.20157 -3.64275 -0.17657 0.21472 0.05646 1.36826 0.71013 -0.79587
##       LeagueN  DivisionW PutOuts Assists   Errors NewLeagueN
## [1,] -0.00569   -0.07798 0.00022 0.00004 -0.00017   -0.00108
## [2,] 48.78548 -120.32652 0.25597 0.13825 -3.45334  -12.72417
## [3,] 63.45466 -117.04221 0.28201 0.37384 -3.42458  -26.00703
```

Clearly, at $\lambda = 10^6$, all the coefficients are shrunk to very nearly zero, at $\lambda = 39$, they are moderately shrunk, and at $\lambda = 0.001$, the values are likely very close to least-square regression coefficients. This follows the trend we expect as we tune λ.

A Quick Plot to Summarize This

```
plot(ridge.mod)
```



From this plot, we can see that as ℓ_2 norm increases (same as λ decreasing), the coefficients move farther away from zero.

Training-Test Split

For reproducibility, we will set a seed. We will then split the dataset into 1/2 training set and 1/2 test set, making different variables for x and y.

```
set.seed(12345)

train_indices = sample(1:nrow(x), nrow(x)/2) #Splits indices into 1/2 training
test_indices = -train_indices #...and 1/2 test

x.train = x[train_indices,]
x.test = x[test_indices,]
y.train = y[train_indices]
y.test = y[test_indices]
```

Finding the Optimal Lambda

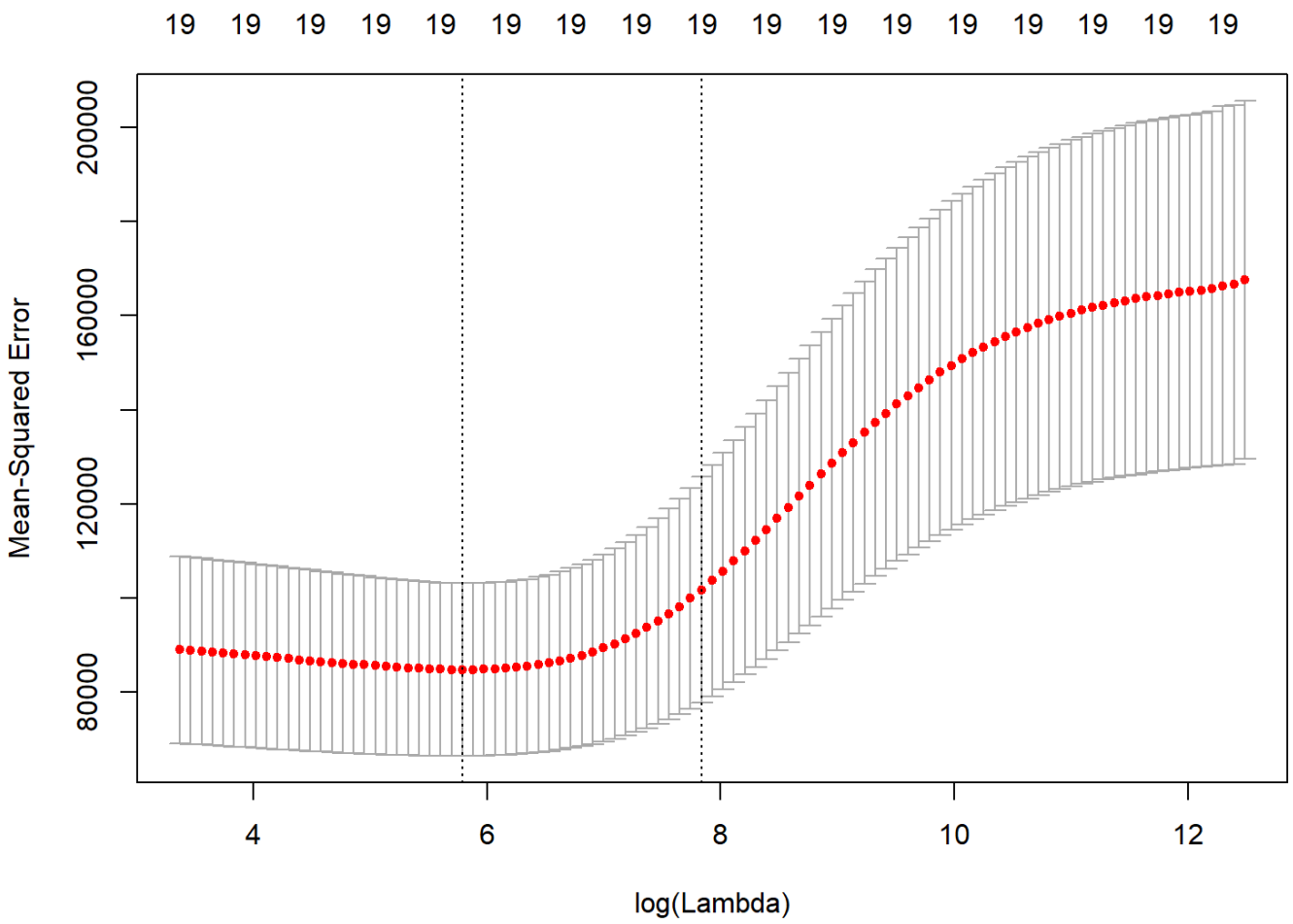
We must use cross-validation on our training set to find an optimal lambda. We can use the `cv.glmnet` function.

```
set.seed(12345)

cv.out = cv.glmnet(x.train, y.train, alpha = 0) #Creates cross-validation ridge model on training set
bestlam = cv.out$lambda.min  #Finds optimal lambda from cross-validation
print(paste("The optimal lambda is:", round(bestlam,2)))
```

```
## [1] "The optimal lambda is: 326.4"
```

```
plot(cv.out)
```



Finding Test MSE with Lambda

We would then use this lambda to find our test MSE, and will compare this to training MSE.

```
#Make predictions on training set and test
ridge.pred1 = predict(ridge.mod, s = bestlam, newx = x.train)
ridge.pred=predict (ridge.mod ,s=bestlam, newx=x.test)
#Finds MSE on training and test set
ridge_test_mse = mean((ridge.pred -y.test)^2)
ridge_train_mse = mean((ridge.pred1 - y.train)^2)
```

```
## [1] "Train MSE is: 70961.64"
```

```
## [1] "Test MSE is: 145381.53"
```

Clearly, the training MSE is significantly lower than test MSE, which is expected since the model was created from the training set.

Since this is a very small dataset, this may not necessarily be our best λ to predict our test set. However, in larger datasets, cross validation will be a lot more likely to find an optimal λ that could assure a low test error.

Final Coefficients

The final coefficients for this model would be:

```
out = glmnet(x,y, alpha = 0)
#Finds coefficients on whole dataset using optimal lambda
predict(out, type = "coefficients", s= bestlam)[1:20,]
```

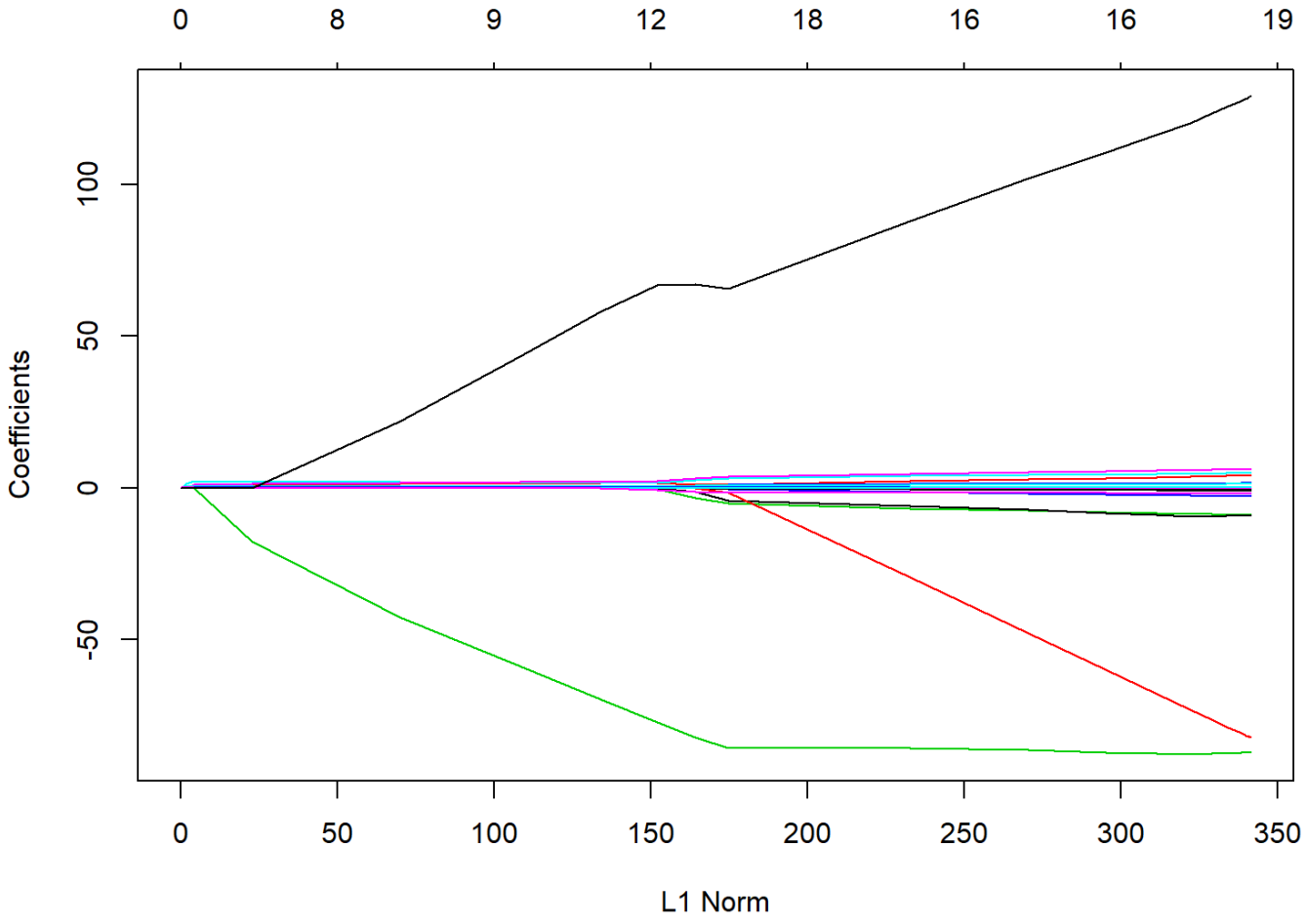
```
##  (Intercept)        AtBat         Hits        HmRun         Runs
##  15.46888688   0.07723297   0.85882036   0.60198653   1.06356145
##          RBI        Walks        Years       CAtBat        CHits
##   0.87935926   1.62405600   1.35484930   0.01134995   0.05745115
##       CHmRun        CRuns         CRBI       CWalks      LeagueN
##   0.40670536   0.11453292   0.12113244   0.05303368  22.08031781
##    DivisionW      PutOuts      Assists       Errors   NewLeagueN
## -79.01029208   0.16613954   0.02939529  -1.35997016   9.12713059
```

LASSO

The process for performing LASSO regression is very similar, as all you have to do is change the `alpha` parameter in the `glmnet` functions to 1. It of couse will have the unique property of shrinking unnecssary variables all the way to zero.

Similar to ridge, we create the model with the same λ grid and show a plot of the ℓ_1 norm with:

```
lasso.mod = glmnet(x.train, y.train, alpha = 1, lambda = grid) #Note alpha = 1
plot(lasso.mod)
```



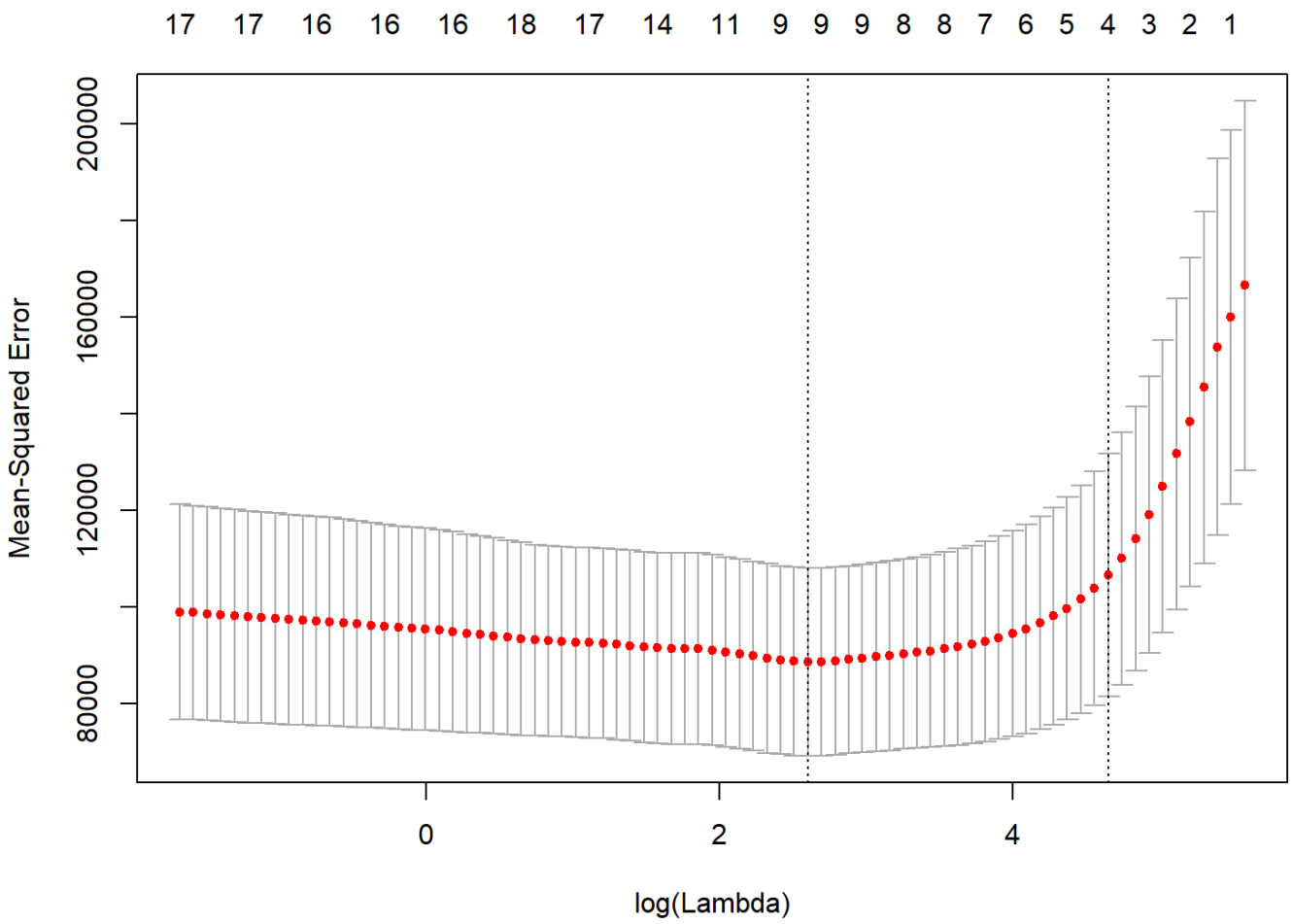
LASSO Cross Validation

Again, similar to ridge, we choose an optimal lambda through cross validation…

```
#Same process as ridge
set.seed(12345)
cv.out = cv.glmnet(x.train, y.train, alpha = 1)
bestlam = cv.out$lambda.min
print(paste("The optimal lambda is:", round(bestlam,2)))
```

```
## [1] "The optimal lambda is: 13.49"
```

```
plot(cv.out)
```



LASSO

And compute the test error and final model coefficients.

```
#Same process as ridge
set.seed(12345)
lasso.pred=predict(lasso.mod ,s=bestlam, newx=x.test)
lasso_test_mse = mean((lasso.pred -y.test)^2)
print(paste("Test MSE is:", round(lasso_test_mse,2)))
```

```
## [1] "Test MSE is: 150801.33"
```

```
out = glmnet(x,y, alpha = 1)
predict(out, type = "coefficients", s= bestlam)[1:20,]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  8.347202e+00  0.000000e+00  1.918988e+00  0.000000e+00  0.000000e+00
##           RBI         Walks         Years        CAtBat         CHits
##  0.000000e+00  2.251773e+00  0.000000e+00  0.000000e+00  0.000000e+00
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##  5.712264e-05  2.097267e-01  4.180275e-01  0.000000e+00  9.319658e+00
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -1.093501e+02  2.280092e-01  0.000000e+00 -8.996060e-02  0.000000e+00
```

This model selects 9 variables and shrinks the rest to zero, with a relatively similar test MSE to the one found in ridge regression.

Effect of a Different Seed

To show how the model can change by a different selection of cross validation, we will run the previous steps on LASSO with a different seed.

```
set.seed(123)
#All the previous code is run again below, not shown
```

```
## [1] "The optimal lambda is: 16.25"
```

```
## [1] "Test MSE is: 150779.48"
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs
##   17.6414447    0.0000000    1.8772483    0.0000000    0.0000000
##          RBI        Walks        Years       CAtBat        CHits
##    0.0000000    2.2244214    0.0000000    0.0000000    0.0000000
##       CHmRun        CRuns         CRBI       CWalks      LeagueN
##    0.0000000    0.2079576    0.4134347    0.0000000    2.9611688
##    DivisionW      PutOuts      Assists       Errors   NewLeagueN
## -104.4544882    0.2218936    0.0000000    0.0000000    0.0000000
```

Although the test error and lambda are relatively similar, the model only selects 7 coefficients instead of 9. This variation would likely decrease using a larger dataset.

Chapter 8: Tree-Based Methods

Ishan Shah

April 13, 2019

Decision Trees

Main Idea:
1. We *stratify or segment* the predictor space into a number of regions.
2. For each observation, determine which segment (region) it belongs to
3. We then are able to make a prediction for a given observation by simply using the mode (classification) or the mean (regression) of all the outcomes in that segment.

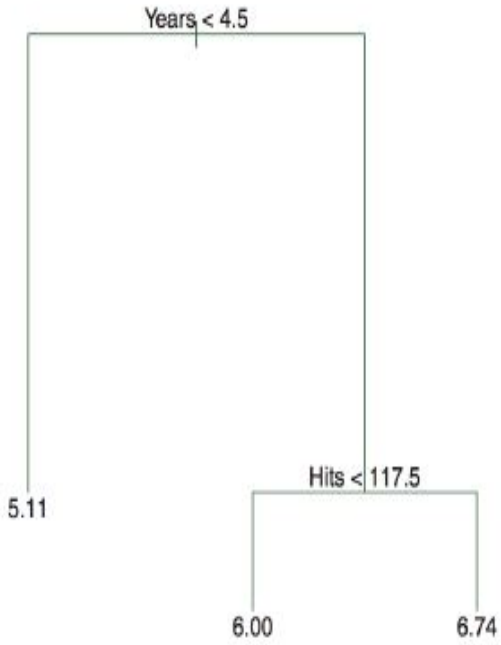
The splitting rules are summarized in trees, making them easily interpretable.

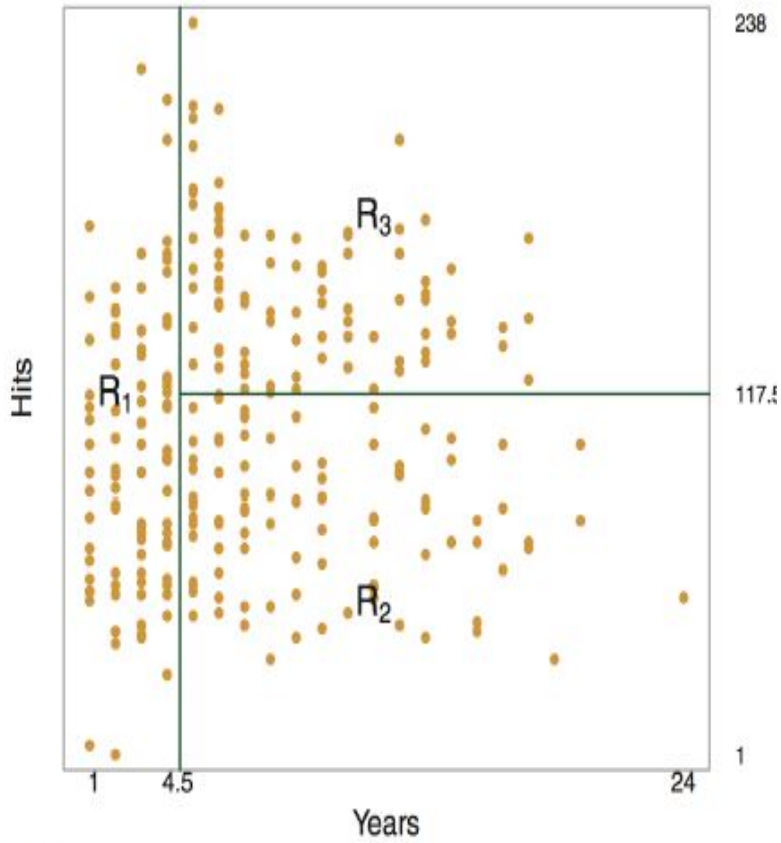
However, single decision trees are not competitive to other supervised methods, so we usually need to use methods that combine trees, such as **bagging, boosting, and random forests**, to produce significantly more accurate predictions at the expense of some interpretability.

Regression Tree Example

Here, we are trying to predict baseball players `Salary` (which is in scale of thousands and log_e), using predictors `Years` (years of experience in MLB) and `Hits` (number of hits made in previous year).

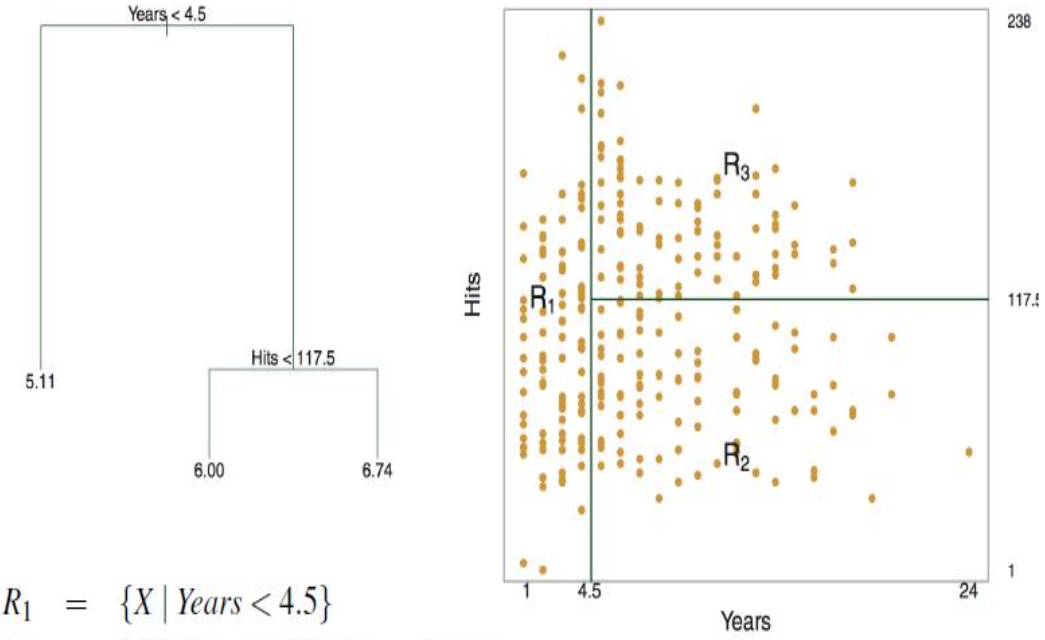
The tree divides the predictor space into **3 regions**.





$$R_1 = \{X \mid Years < 4.5\}$$
$$R_2 = \{X \mid Years \geq 4.5, Hits < 117.5\}$$
$$R_3 = \{X \mid Years \geq 4.5, Hits \geq 117.5\}$$

Interpretation



$$R_1 = \{X \mid Years < 4.5\}$$
$$R_2 = \{X \mid Years \geq 4.5, Hits < 117.5\}$$
$$R_3 = \{X \mid Years \geq 4.5, Hits \geq 117.5\}$$

For players with less than 4.5 years experience, their predicted salary is $1000 * e^{5.11}$ dollars (the mean salary for such players in training data).

For players with more than 4.5 years experience, we must also consider the number of hits. If they have < 117.5 hits, predicted salary is $1000 * e^{6.00}$ dollars and with > 117.5 hits, $1000 * e^{6.74}$ dollars.

This is likely an oversimplification of the true relationship between these three variables, but is very easily interpretable.

Partitioning Predictor Space

Let's assume the number of regions we divide the predictor space on J is known. How could we find regions $R_1, \ldots, R_j$?

We must divide the space into high-dimensional rectangles (or boxes) that minimize the RSS

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y_{R_j}})^2$$

However, we cannot consider every possible partition so we have to use a top-down, greedy approach called **recursive binary splitting**

- The top-down component similar to hierarchical clustering
- Greedy component means the best split is determined at that particular step, instead of considering best global step

Recursive Binary Splitting

We must select the predictor X_j and cutpoint s such athat splitting the predictor space into regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to greatest possible reduction in RSS.
Considering all predictors $X_1, \ldots, X_p$ and all possible values of cutpoint s, this algorithm can be solved relatively quickly, especially when p is low.

We continue splitting the regions with the highest RSS reduction until an ending criteria, which, for example, could be that there are a maximum of five observations in each region. At the end, we have regions $R_1, \ldots, R_j$.

Note: We can split using the same predictor multiple times!

Pruning the Tree

Problem: The number of regions J determines the complexity of the tree. A complex tree with higher J will predict the training data well but may overfit the data and lead to a high test error. We need a way to **reduce the variance** of a complex tree at the cost of a little bias.

Solution: We must create a smaller tree. The best way is to run recursive binary splitting until we have a large tree, then **prune** it to obtain subtree.

We need a subtree that outputs the lowest test error, but it is too computationally expensive to test every possible subtree using cross-validation.

Pruning the Tree

We have to go back to the **bias-variance tradeoff** here.

We have to prune the tree so it has fewer splits, essentially decreasing the variance of the tree at the cost of a little bias. This will make the tree more flexible to test new datasets.

Since we cannot test every subtree, we use a method called **cost complexity pruning**, where we have α as a tuning parameter. α controls the amount of leaves a subtree $T \subset T_0$, where T_0 is the full tree, in this equation:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - y_{\hat{R}_m})^2 + \alpha |T|$$

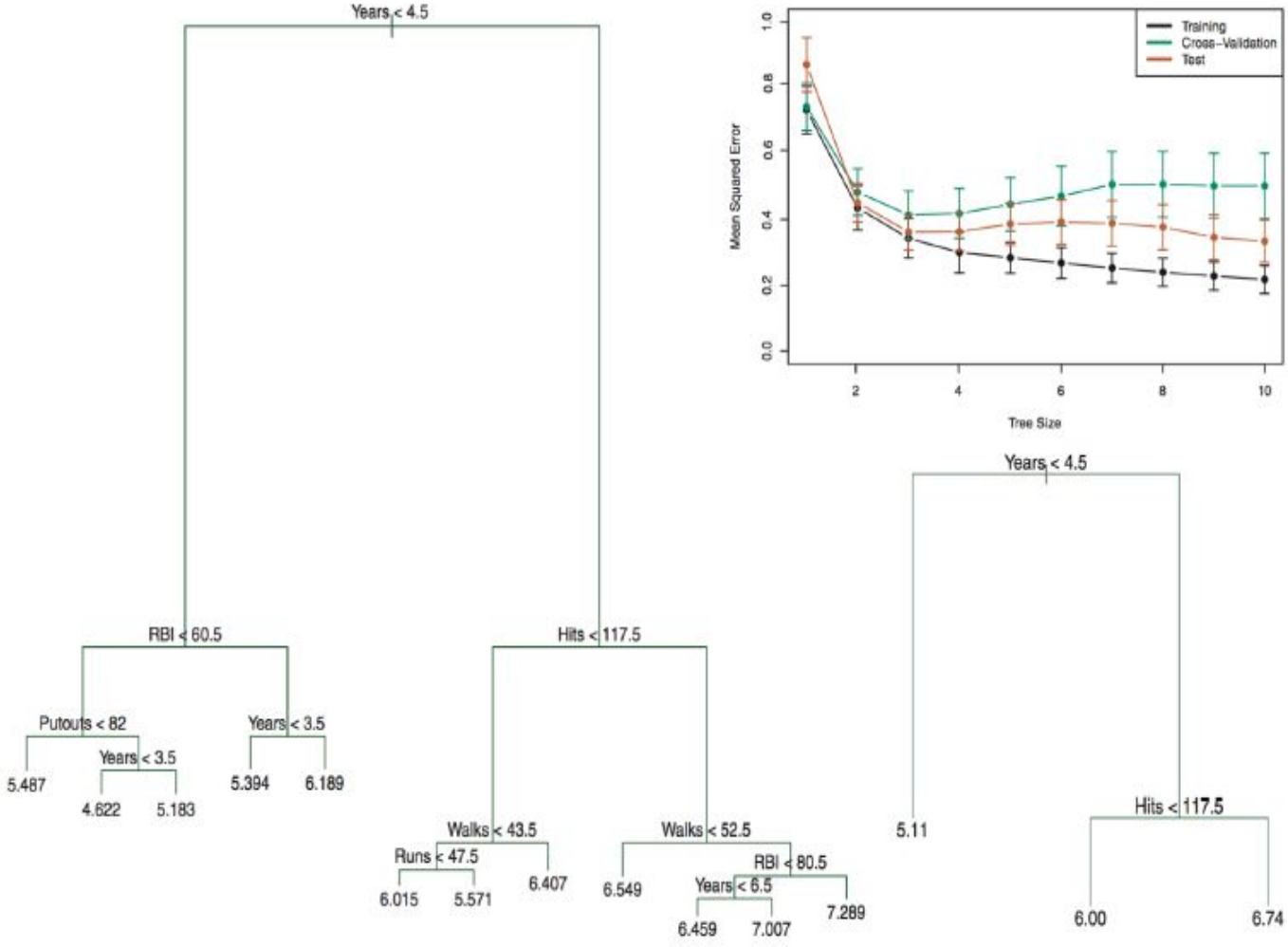
$|T|$ is the amount of terminal nodes in the tree T. This equation resembles LASSO regression.

$\alpha = 0$ is where the subtree T is equal to full tree T_0, and the equation calculates training error. As α increases, the subtree will become smaller, thus decreasing variance and increasing bias.

And just like ridge and LASSO regression, we use **k-fold cross validation** with a grid of α values in order to find the appropriate subtree that minimizes bias and variance.

Pruning the Tree Visualized

This is a visualization of the full decision tree for `Hitters` data and how it was pruned to just 3 nodes. The graph indicates CV error is at a minimum at a subtree with size 3.



Classification Trees

The previous examples were **regression trees**, where a numerical response was predicted. **Classification trees** instead predict a qualitative response.

Once we have assigned a new observation to a given region from the training set, its predicted response is the *most commonly occurring class*, also known as *the majority vote, or the mode* in the region. We are not only interested in the predicted class, but also the proportion of each class for a given region.

Growing Classification Trees

We use recursive binary splitting like regression trees, but use different error metrics to grow the tree.

Instead of using basic classification error, we want to maximize **node purity**, which essentially means we want each node (region) to have a very high proportion of a single class, meaning we have more confidence in the class assignment if an observation is predicted to this node.

To maximize node purity, we grow the tree by **minimizing** either of these two values:

The Gini index:

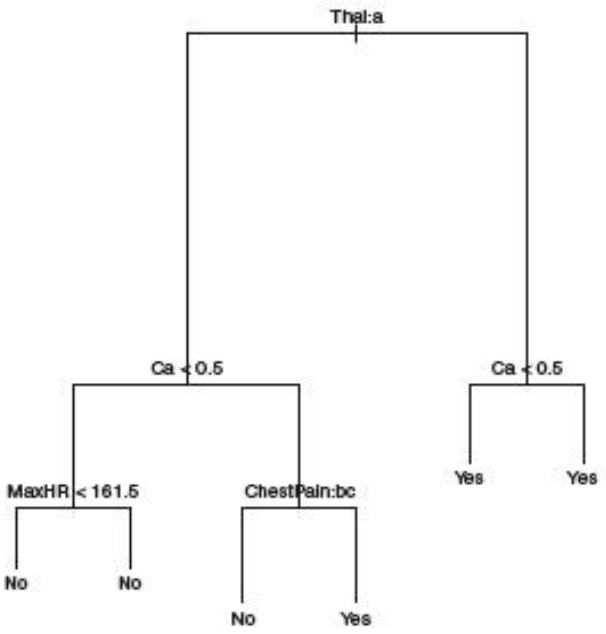
$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

The cross-entropy:

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

where $\hat{p}_{mk}$ is the proportion of training observations in the mth region from the kth class. Both these values are low when $\hat{p}_{mk}$ are close to zero or one, indicating high node purity.

Interpretation



This is a classification tree created on the dataset `Heart`, predicting a binary outcome HD (whether a patient has heart disease or not) based on variables including `Thal` (qualitative thailum stress test), `ChestPain` (qualitative), and `Chol` (a quantitative cholesterol measurement).

The first branch says move to the left if the variable `Thal` takes its first value (indicated by the "a"), and right otherwise. We keep moving to the left if the conditions at the branch are met, and right otherwise, until we reach a terminal node where we can predict the outcome.

Some branches have terminal nodes with the same outcome. This is because the node purity is high in one and low in the other. In the rightmost branch, the left node has high purity, (9/9 observations being "Yes"), indicating high certainty of prediction, and vice versa for the right node (7/11 observations being "Yes").

Interpretation cont.

Like the regression tree, this shows where the CV error is minimized and the full vs. pruned tree.

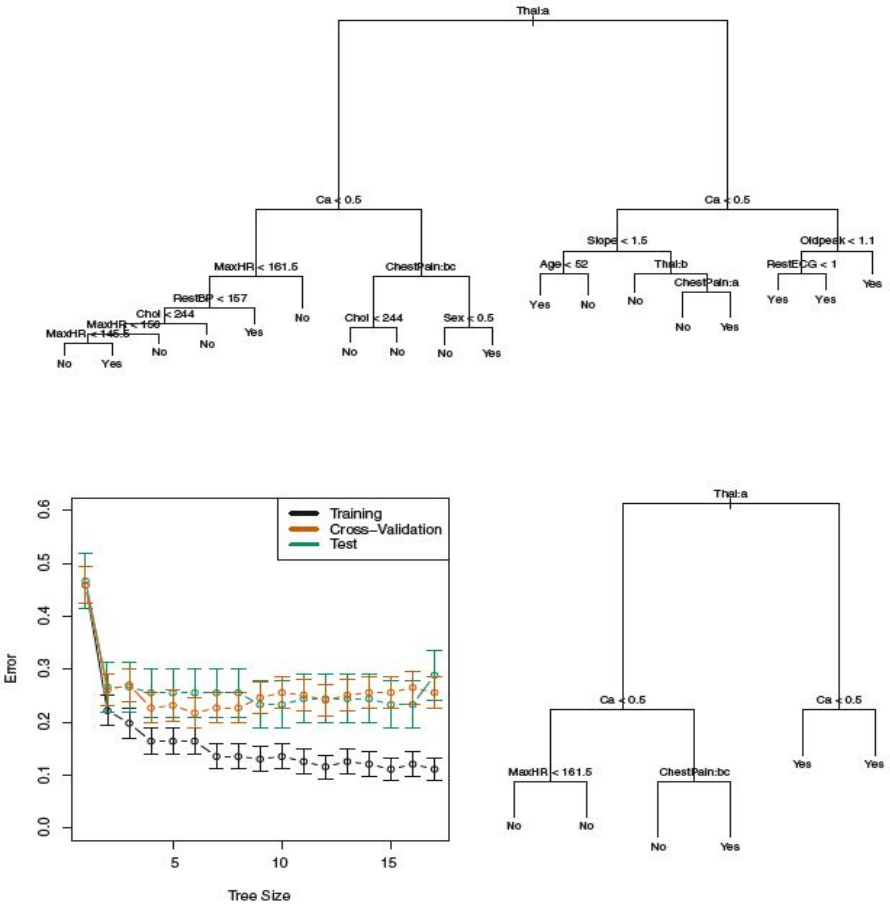


FIGURE 8.6. `Heart` *data.* Top: *The unpruned tree.* Bottom Left: *Cross -validation error, training, and test error, for different sizes of the pruned tree.* Bottom Right: *The pruned tree corresponding to the minimal cross-validation error.*

Linear vs Nonlinear Decision Boundary

Often, trees do better than regression at capturing variability decided by a nonlinaear boundary, shown below.

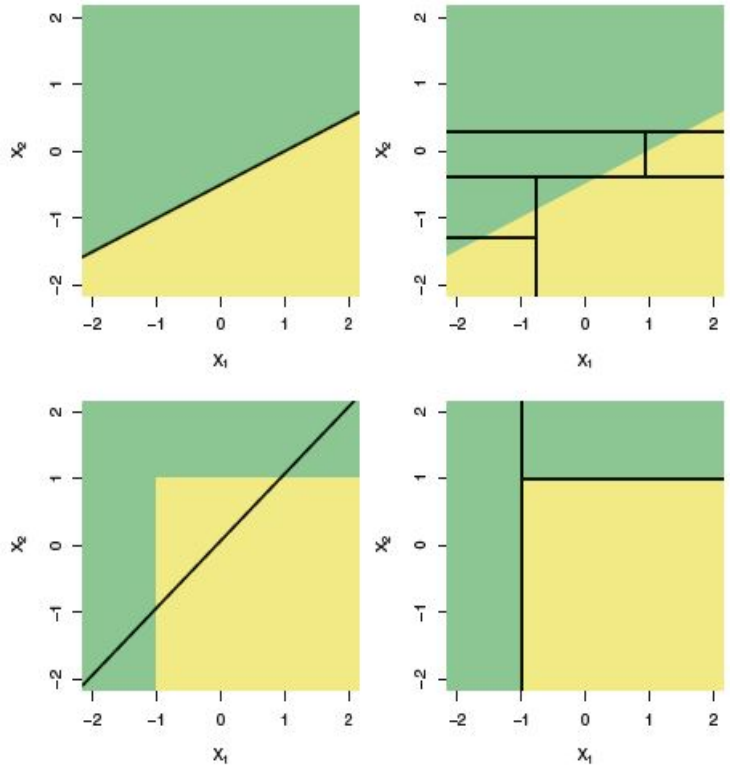


FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).* Bottom Row: *Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).*

Bagging

Bagging, also known as bootstrap aggregation, is, in general, a way to **reduce variance** (increase the model's flexibility) of a statistical learning method. This method is particularly useful in decision trees, which by itself often lacks good prediction performance.

Method:
We essentially generate B training sets using the bootstrap method, and create full (un-pruned) decision trees for each. We then average the B trees to reduce the high variance of individual trees.

Performace Measure: We use **Out-of-Bag (OOB) Error Estimation**. About 2/3 of the full data's observations are used to create each tree, so each observation is not used in about $B/3$ trees. We average (regression tree), or take majority vote (classification tree) of the predicted responses for each of the n observations in the $B/3$ trees it is NOT used in. This gives an OOB error for each observation and, when averaged, gives an estimate of test error.

Pros and Cons: Successfully reduces variance of decision trees and gives more accurate predictions, but at the cost of the normally very easy interpretability of the decision trees.

Random Forests

Random Forests is simply a modification of bagging, where we try to **decorrelate** all the trees created in order to further reduce variance and gain better predictions.

Method: When the decision trees are being created, at each split, a random sample of m predictors (most commonly, $m \approx \sqrt{p}$) are taken from the p total predictors. ONLY these m predictors are considered at the split, preventing the most powerful predictors from having too much influence in creating the tree, thus making the average of all the trees less variable.

Performance Measure:
OOB Error, same as bagging.

Pros and Cons: Often improves prediction performance over bagging by decreasing variance of trees, but the correct rule for choosing m must be decided.

Bagging vs. Random Forest Performance

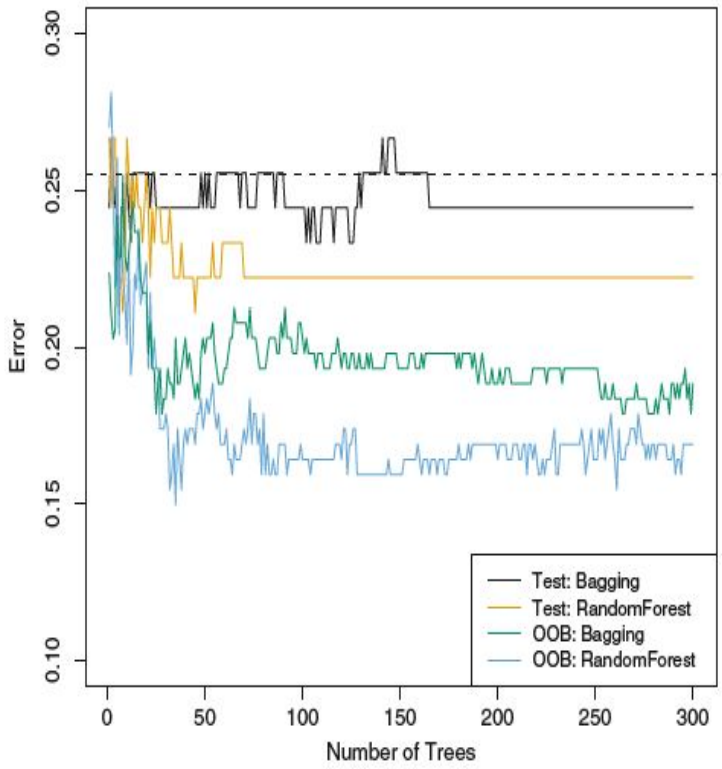


FIGURE 8.8. *Bagging and random forest results for the* `Heart` *data. The test error (black and orange) is shown as a function of* B*, the number of bootstrapped training sets used. Random forests were applied with* $m = \sqrt{p}$*. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is considerably lower.*

Here, Random Forests are seen to outperform bagging.

Boosting

Boosting involves decision trees being grown sequentially based on information from previously grown trees. Each tree is fit on a modified version of the original dataset.

Method: We fit trees to the residuals from the previous model as the response. We then add this new decision tree to the fitted function and update the residuals. This is done using 3 tuning parameters:

Number of Trees B: use cross-validation to select, can overfit if too large.

Shrinkage parameter λ: Small positive number (like 0.01 or 0.001) that controls rate of learning, a smaller number allowing more different shaped trees to model residuals. Small λ needs high B.

Number of splits in each tree d: controls complexity of boosted tree, often $d = 1$ works well.

Performance Measure:
Test error using different values of tuning parameters.

Pros and Cons:
Often more interpretability than previous methods, but can be slow.

Recommended Problems

8.5, 8.8, 8.9, 8.10

ISLR Chapter 8 Problem 8

Ishan Shah

April 22, 2019

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.5.2
```

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 3.5.3
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.5.3
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
```

In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable. Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable.

Summary of Carseats dataset

```
summary(Carseats)
```

```
##      Sales          CompPrice       Income        Advertising
##  Min.   : 0.000   Min.   : 77   Min.   : 21.00   Min.   : 0.000
##  1st Qu.: 5.390   1st Qu.:115   1st Qu.: 42.75   1st Qu.: 0.000
##  Median : 7.490   Median :125   Median : 69.00   Median : 5.000
##  Mean   : 7.496   Mean   :125   Mean   : 68.66   Mean   : 6.635
##  3rd Qu.: 9.320   3rd Qu.:135   3rd Qu.: 91.00   3rd Qu.:12.000
##  Max.   :16.270   Max.   :175   Max.   :120.00   Max.   :29.000
##    Population        Price        ShelveLoc        Age
##  Min.   : 10.0   Min.   : 24.0   Bad   : 96   Min.   :25.00
##  1st Qu.:139.0   1st Qu.:100.0   Good  : 85   1st Qu.:39.75
##  Median :272.0   Median :117.0   Medium:219   Median :54.50
##  Mean   :264.8   Mean   :115.8                Mean   :53.32
##  3rd Qu.:398.5   3rd Qu.:131.0                3rd Qu.:66.00
##  Max.   :509.0   Max.   :191.0                Max.   :80.00
##    Education    Urban       US
##  Min.   :10.0   No :118   No :142
##  1st Qu.:12.0   Yes:282   Yes:258
##  Median :14.0
##  Mean   :13.9
##  3rd Qu.:16.0
##  Max.   :18.0
```

a. Split the data into training and test set

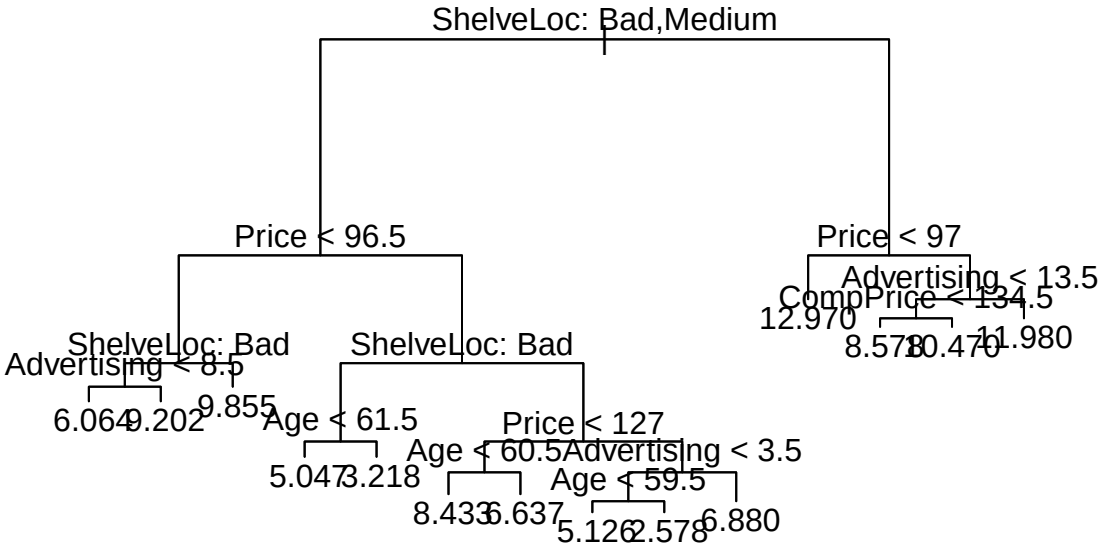
```
set.seed(27514) #change this and see what happens
train = sample(1:nrow(Carseats), (nrow(Carseats)/2))#split into 1/2 train, 1/2 test
Car.train = Carseats[train, ] #Training set
Car.test = Carseats[-train,] #Test set
```

b. Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

```
reg.tree = tree(Sales~.,data = Car.train) #Create the tree based on all predictors
summary(reg.tree)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = Car.train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Advertising" "Age"         "CompPrice"
## Number of terminal nodes:  14
## Residual mean deviance:  2.396 = 445.7 / 186
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.74300 -1.15300 -0.06233  0.00000  0.90020  5.09300
```

```
plot(reg.tree) #Plots the tree below
text(reg.tree, pretty = 0)
```



The tree has 14 terminal nodes. Let's see the test MSE that results from it.

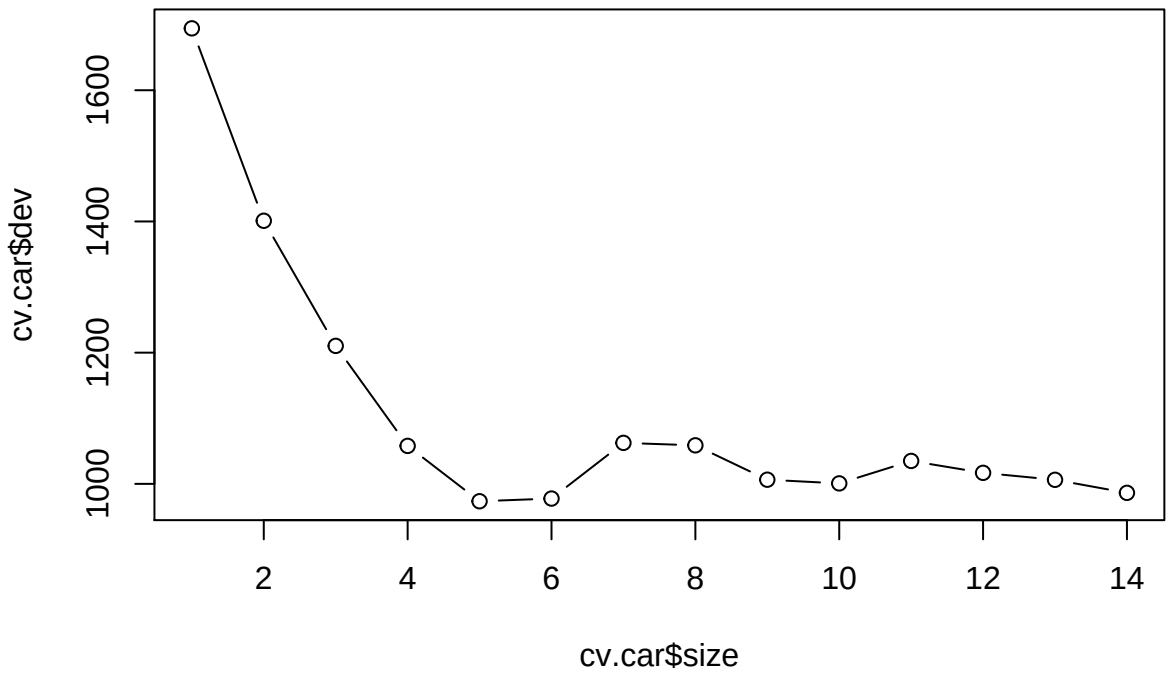
```
yhat = predict(reg.tree,newdata = Car.test)
mean((yhat - Car.test$Sales)^2) #find MSE on test set
```

```
## [1] 4.75955
```

The test MSE here is 4.76.

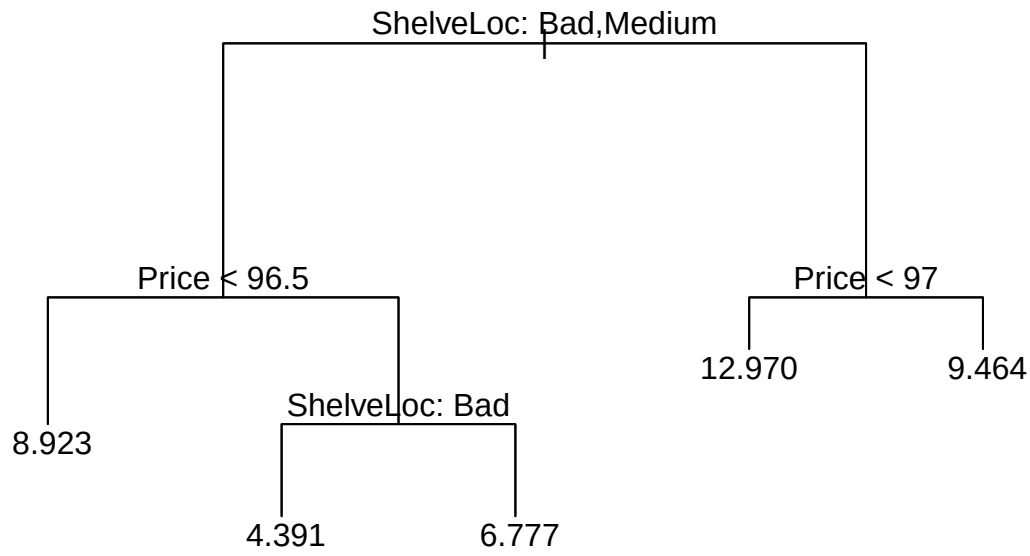
c. Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

```
set.seed(27514)
cv.car = cv.tree(reg.tree) #Runs cross validation on different sizes of the tree and outputs error at e
plot(cv.car$size, cv.car$dev, type = "b")
```



This plots the cross-validation error rate on the y axis vs. the size of the tree on the x axis. We want to choose the tree size that produces the lowest cross-validation error, and use that tree on the test set. We see here a minimum is reached at 5 nodes, so we will attempt to prune the tree to 5 nodes.

```
prune.car = prune.tree(reg.tree, best = 5)
plot(prune.car)
text(prune.car,pretty=0)
```




```
yhat=predict(prune.car, newdata= Car.test)
mean((yhat-Car.test$Sales)^2)
```

```
## [1] 5.029553
```

In this case, the pruned tree **DID NOT** produce a lower test MSE than the full tree. This is likely because it is being fit on a very small dataset. Try this with more data and the large try would likely overfit and produce bad test results.

d. Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important.

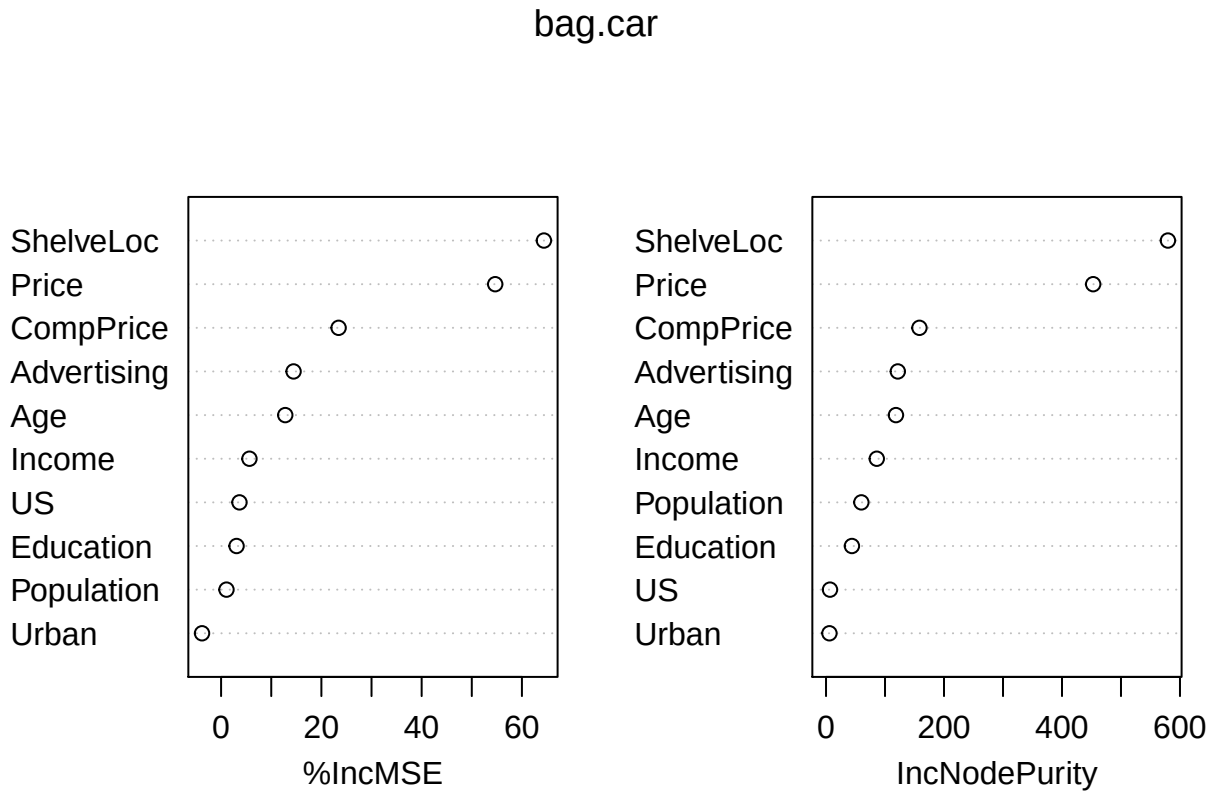
```
set.seed(27514)
bag.car = randomForest(Sales~.,data=Car.train,mtry = 10, importance = TRUE) #mtry is the number of vari


yhat.bag = predict(bag.car,newdata=Car.test)
mean((yhat.bag-Car.test$Sales)^2)
```

```
## [1] 2.64321
```

The test MSE of 2.64 is significantly lower than what we saw when doing a single regression tree. This shows how combining many trees can increase prediction power.

```
varImpPlot(bag.car)
```



We can see based on these graphs that the most important variables by far are price and quality of shelf location, as they lead to the highest increases in Node Purity and have the most influence on the MSE

e. Use random forests to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

```
set.seed(27514)
rf.car = randomForest(Sales~.,data=Car.train,mtry = 3, importance = TRUE) #we use m = sqrt(p) here, mea
yhat.rf = predict(rf.car,newdata=Car.test)
mean((yhat.rf-Car.test$Sales)^2)
```

```
## [1] 2.989369
```

Here, randomForest has a higher MSE than bagging, not improving on it. This likely means the variables were already not highly correlated so attempting to use less predictors at each split didn't actually change the results much. Again, this could vary greatly with a larger dataset and a different seed.

10.2: Principal Components Analysis

Ishan Shah

February 23, 2019

Unsupervised Learning

Principal Components Analysis (PCA) is a form of **Unsupervised Learning**.

In unsupervised learning, there is no Y, we simply try and draw interesting insights from measurements on $X_1, X_2, \ldots, X_p$. It is used more for data visualization, preprocessing, and exploratory analysis than for predictive modeling.

Challenge:

Significantly more **subjective** than supervised learning - there often isn't a clear way to assess quality of results because we can't check our work with a test set.

However, it has significant utility in various fields.

Examples:

How can unique subgroups among genetic data give us a better understanding of a disease?

How can an online shopping site target marketing from identifying unique groups of shoppers with similar browsing patterns?

Principal Components

Big Idea: When we have many correlated variables, it is nearly impossible to visualize their relationships individually through simple 2D scatterplots. What is a better way to do this?

We can summarize all these variables in a smaller number of representative variables, known as **principal components**, that still capture the majority of the data's variability. This is a form of **dimension reduction**.

Definition of PCA: The process by which principal components are computed, and the subsequent use of these components to understand the data.

Each principal component is a linear combination of the data's p predictors, essentially a weighted average. There can be at max $min(n-1, p)$ principal components, where n is the amount of rows in the dataset.

Principal Components

The **first** principal component of $X_1, X_2, \ldots, X_p$ is defined by a normalized linear combination:

$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \ldots + \phi_{p1}X_p$, where

$$\sum_{j=1}^{p} \phi_{j1}^2 = 1$$

is the normalization, and…

$\phi_1 = (\phi_{11}, \phi_{21}, \ldots, \phi_{p1})^T$ is the first principal component **loading vector**, whose sum of squares equals one. Essentially, each predictor contributes one value in each of the loading vectors $\phi_1, \phi_2, \ldots$.

How do we find the first component?
We have to assume the variables are centered with mean zero, meaning its important to **standardize** the variables before running a PCA.

We find the value:

$z_{i1} = \phi_{11}x_{i1} + \phi_{21}x_{i2} + \ldots + \phi_{p1}x_{ip}$

that has the largest sample variance, subject to the normalization constraint for ϕ values. This is done through an *eigenvalue decomposition* (outside the scope of this course).

where $z_{11}, z_{21}, \ldots, z_{n1}$ are the **principal component scores** for principal component 1,

And then...

To find the second component and so on, we use the same process, but add an additional constraint that Z_2 has to be completely uncorrelated with Z_1, to assure it doesn't account for any of the variance already captured in Z_1.
Follwing this, all the principal components have to be uncorrelated with each other.

Loading Factors Interpretation

- Coefficients with similar loading factors (LFs) within a PC tend to be directly correlated
- As the absolute value of LFs get closer to 1, the coefficients are more influential on the PC

	PC1	PC2
Murder	0.5358995	−0.4181809
Assault	0.5831836	−0.1879856
UrbanPop	0.2781909	0.8728062
Rape	0.5434321	0.1673186

TABLE 10.1. *The principal component loading vectors, ϕ_1 and ϕ_2, for the* USArrests *data. These are also displayed in Figure 10.1.*

Conclusions from Loading Factors:

Murder, assault, and rape, seem to be very positively correlated due to their similar values.

These three are also very influential as their sum of squares comprise about 92% of the ϕ_1 Loading vector ($\frac{(.535)^2+(.583)^2+(.543)^2}{1} = 0.92$)

Urban population makes up the vast majority of loading vector ϕ_2, and seems to be the most inversely correlated with murder.

Geometric Interpretation

The previous slide may be difficult to picture, but there is a nice geometric interpretation for principal components that can help validate the loading factors.

Loading vector ϕ_1 defines a **direction** on predictor space where the data has highest variance. After projecting n data points on this direction, we obtain principal component scores $z_{11}, \ldots, z_{n1}$ and can plot them. Due to it being completely uncorrelated, second loading vector ϕ_2's direction will be orthogonal to that of the first vector.

Geometric Interpretation

Looking back at our USArrests Data:

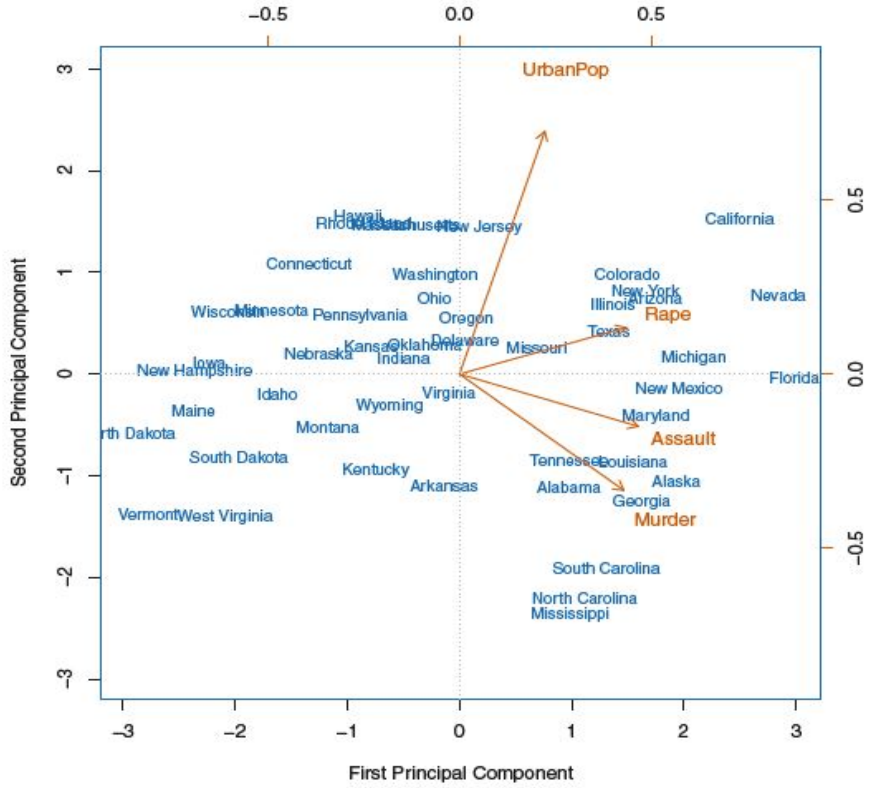


FIGURE 10.1. *The first two principal components for the* USArrests *data. The blue state names represent the scores for the first two principal components. The orange arrows indicate the first two principal component loading vectors (with axes on the top and right). For example, the loading for* Rape *on the first component is* 0.54*, and its loading on the second principal component* 0.17 *(the word* Rape *is centered at the point* $(0.54, 0.17)$*). This figure is known as a biplot, because it displays both the principal component scores and the principal component loadings.*

Note: This plot uses the same scale for the first 2 principal components. Since the first principal component explains significantly more of the data's variance than the second, we should picture the x axis to be signficantly wider than it is now.

- PC1 emphasizes crime variables, which are strongly correlated due to their similar directions
- PC2 emphasizes urban population, whose direction is farthest away from the murder variable

- Ex: California has high crime and high urban population whereas Vermont has low values of both

Another Geometric Interpretation

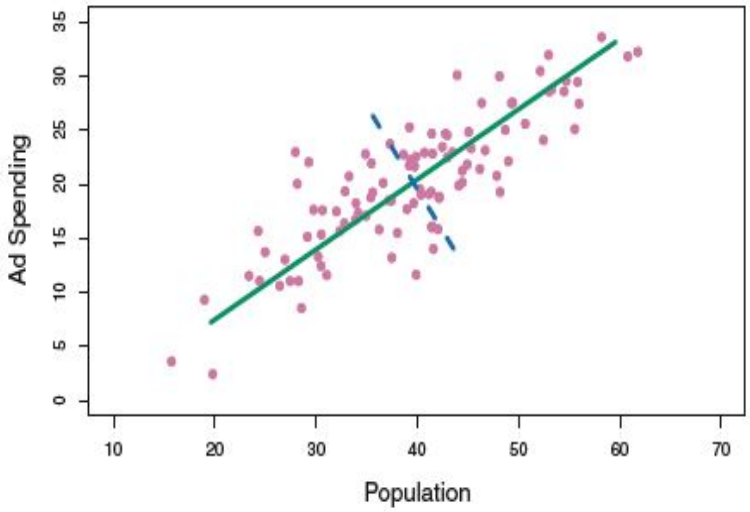


FIGURE 6.14. *The population size (pop) and ad spending (ad) for 100 different cities are shown as purple circles. The green solid line indicates the first principal component, and the blue dashed line indicates the second principal component.*

Another, possibly simpler, way to interpret this concept is that **principal components are low-dimensional linear surfaces that are closest to observations**.

Essentially, if the line is as close as possible to all the data points, it will likely summarize the data well.

In the figure above, we see that the first PC falls very close to the data points, and the second PC is an orthogonal line to the first PC that fits the data as close as possible.

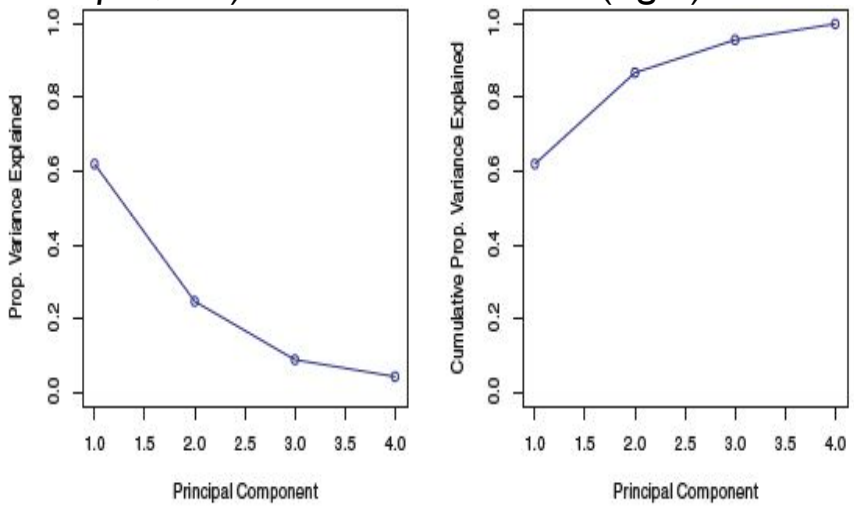
Proportion of Variance Explained (PVE)

PVE, for each of m PCs, simply quantifies how much of the data's variance is explained by that specific PC. The first component will always have the greatest PVE, and the value will continue to decrease in further PCs.

Formally, for the mth component, PVE is (assuming centered, mean zero, variables):

$$\frac{\frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{p}\phi_{jm}x_{ij})^2}{\frac{1}{n}\sum_{j=1}^{p}\sum_{i=1}^{n}x_{ij}^2} = \frac{\sum_{i=1}^{n}z_{im}^2}{\sum_{j=1}^{p}\sum_{i=1}^{n}x_{ij}^2} = \frac{\text{Variance Explained by PC m}}{\text{Data's Overall Variance}}$$

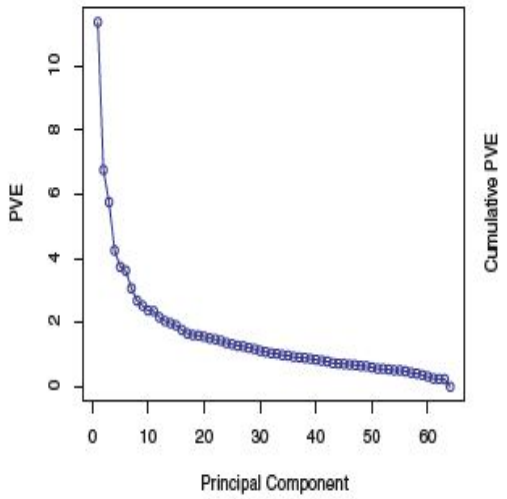
Plots for the `USArrests` data are shown below, displaying the PVE (known as *scree plot*, left) and cumulative PVE (right) at each component:



As you can see, the PVE is very high at the first component (greater than 60%), meaning the three crime variables have very strong influence. The PVE then decreases exponentially for the following PCs.

How many PCs should I use?

In general, there is no formal way to choose the amount of principal components you want to use. Simply put, we want to use the smallest number of PCs (for model simplification) that explain the vast majority of the data's variation. Visually this can be found at the **elbow** on a **scree plot**.

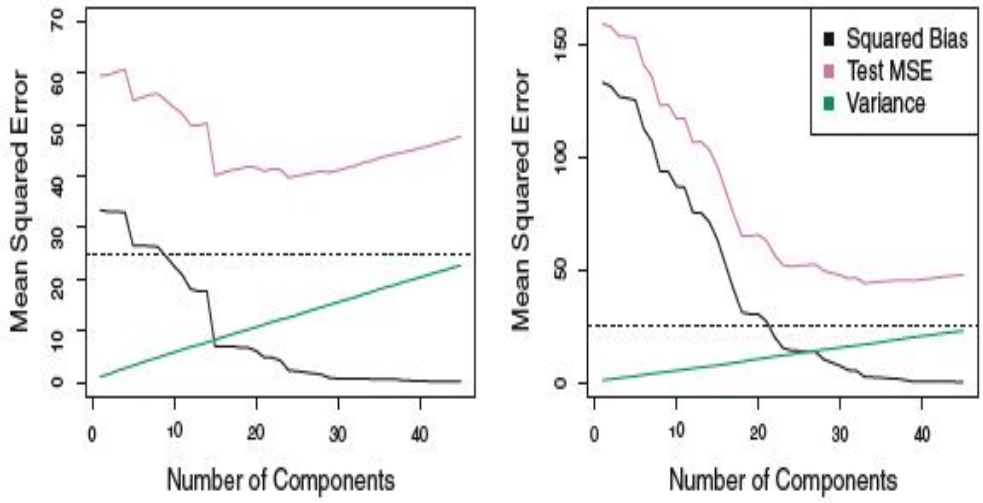


With this dataset of cancer microarray data, we can see that after the **7th** principal component (approx elbow point), each PC explains only a minute amount of the data's variation, not adding much to our analysis.
Consequently, 7 PCs may be an appropriate amount to choose when doing analysis.

Principal Component Regression (6.3)

Principal components can also be used in supervised learning.
Key issue: When $p > n$, least-squares regression does not work and we need to reduce dimensions in order to run a regression. **LASSO** is a way to reduce dimensions, but so are principal components.

To mitigate this issue, we will use a subset of principal components $Z_1, Z_2, \ldots, Z_m$ as our predictors and fit a least squares model. To choose the amount of PCs we want to use, we implement cross-validation to find the optimal amount of principal components that minimize our test error. Plots of this are below on two simulated datasets:



We care less about minimizing the amount of PCs, as all we hope for is good prediction performance. From these graphs, we can see the optimal amount of components is about 20 (left) and 30 (right). **This is a technique that would be much more useful simply for prediction performance. There is hardly any interpretability that arises from least-squares coefficients in these models.**

In Summary

- PCA finds a low dimensional representation of the dataset that contains as much variation as possible
- The first principal component is the line in p dimensional space closest to the observations
- PCA is a useful exploratory analysis tool, particularly through visualizations
- The proportion of variance explained decreases for each PC as more are added, but the cumulative PVE increases
- One can use a scree plot to find the appropriate number of PCs to use
- Principal Components Regression is a useful supervised learning tool for high-dimensional datasets, but mainly for prediction performance as there is little interpretability

10.3: Clustering Methods

Ishan Shah

March 16, 2019

Unsupervised Learning

Clustering is a form of **Unsupervised Learning**.

In unsupervised learning, there is no Y, we simply try and draw interesting insights from measurements on $X_1, X_2, \ldots, X_p$. It is used more for data visualization, preprocessing, and exploratory analysis than for predictive modeling.

Challenge:

Significantly more **subjective** than supervised learning - there often isn't a clear way to assess quality of results because we can't check our work with a test set.

However, it has significant utility in various fields.

Clustering

The point of **clustering** is to find homogeneous subgroups among the dataset's n observations on the basis of its p predictors,resulting in observations in the same group being very similar and observations in different groups being very dissimilar.

Examples:

How can we discover unique subgroups among genetic data that give us a better understanding of a disease?

How can an online shopping site target market from identifying unique groups of shoppers with similar browsing patterns? (*market segmentation*)

The two main methods we will cover are **K-means clustering** and **hierarchical clustering**.

K-means Clustering

Goal: For a fixed preset amount of clusters K, we must assign each observation to exactly one cluster. If the ith observation is in the kth cluster, then $i \in C_k$.

Method: We need to minimize the **within-cluster variation (WCV)** as much as possible for all clusters. This is most often defined by **squared Euclidean distance**

Formally:

$$WCV(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (X_{ij} - X_{i'j})^2$$

Essentially, the summing over i and i' denotes the addition of **all** the pairwise squared euclidean distances between observations in cluster C_k, dividing by $|C_k|$, the number of observations in the cluster. This results in the average distance between two points in the cluster.

Then, we minimize this over all clusters, shown below:

$$\min_{C_1, C_2, \ldots, C_k} \sum_{k=1}^{K} WCV(C_k)$$

Algorithm to Solve K-means

There is no way to find the **global optimum** to the minimization problem from the previous slide, but this algorithm is an efficient way, even on large datasets, to find a **local optimum** for the problem. This is an iterative process, as shown below by step 3:

1. Randomly assign each observation to a cluster $1, \ldots, K$.

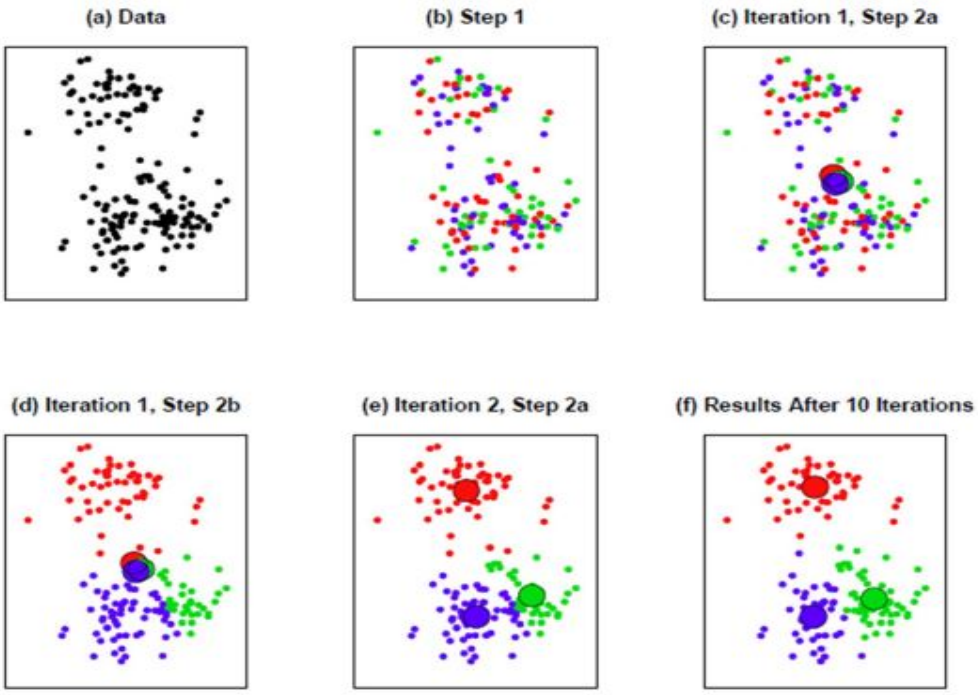
2a. For each of the K clusters, compute the cluster centroid. The kth cluster centroid is the vector of the means of the p predictors for all the observations in the kth cluster (essentially the cluster's midpoint).

2b. Assign each observation to the cluster whose centroid is closest (by Euclidean distance).

3. Repeat steps 2a and 2b until the assignments do not change anymore.

K-means clustering should be run with multiple random starting points, and the starting points with the lowest objective function should be chosen.

K-means Visualization



Clearly, after many iterations, K-means clustering converges to a single solution.

Choosing K in K-means

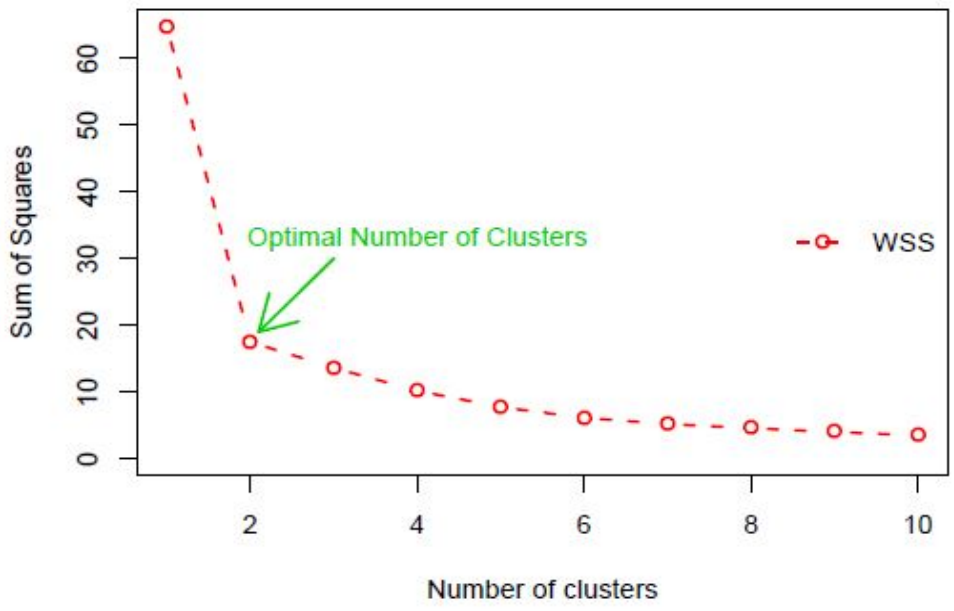
The most common way to choose the amount of clusters for K-means clustering is to test the **Within-Sum-of-Squares (WSS)** for various values of K, and plot this.

WSS is defined as

$$\sum_{k=1}^{K} \sum_{i \in C_k} d^2(m_k; X_i)$$

, where m_k is the centroid of cluster k. It is essentially the sum of the squared distances between all the observations and their cluster centroids.

However, as K increases, WSS usually decreases. Therefore, we need to find a compromise in the value of K, such that there aren't too many clusters but there is also a low WSS. This is found by the plot's **elbow**. After the elbow, the WSS only decreases slightly, and more clusters may not necessarily be useful.



Hierarchical Clustering

The 2 types of hierarchical clustering are **bottom-up** and **top-down**. We will focus on bottom-up (agglomerative) methods as they are significantly more common.

In **Bottom-Up** clustering, we start from n individual clusters and group them together using a similarity measure. We continually merge the closest pair of clusters until there are two clusters.

We can visualize this in am upside down tree structure, starting with leaves and combining clusters up to the trunk; this is known as a **dendogram**.

Similarity Measures

Euclidean Distance:

$$d(i,j) = \sqrt{\sum_{k=1}^{p}(X_{ik} - X_{jk})^2}$$

Manhattan Distance:

$$d(i,j) = \sum_{k=1}^{p}|X_{ik} - X_{jk}|$$

Correlation-Based Distance: Observations are more *similar* if correlation between their features are higher.

In our examples, we will mainly be using **Euclidean Distance**.

Hierarchical Clustering Algorithm

1. Treat each of the n observations as its own cluster.

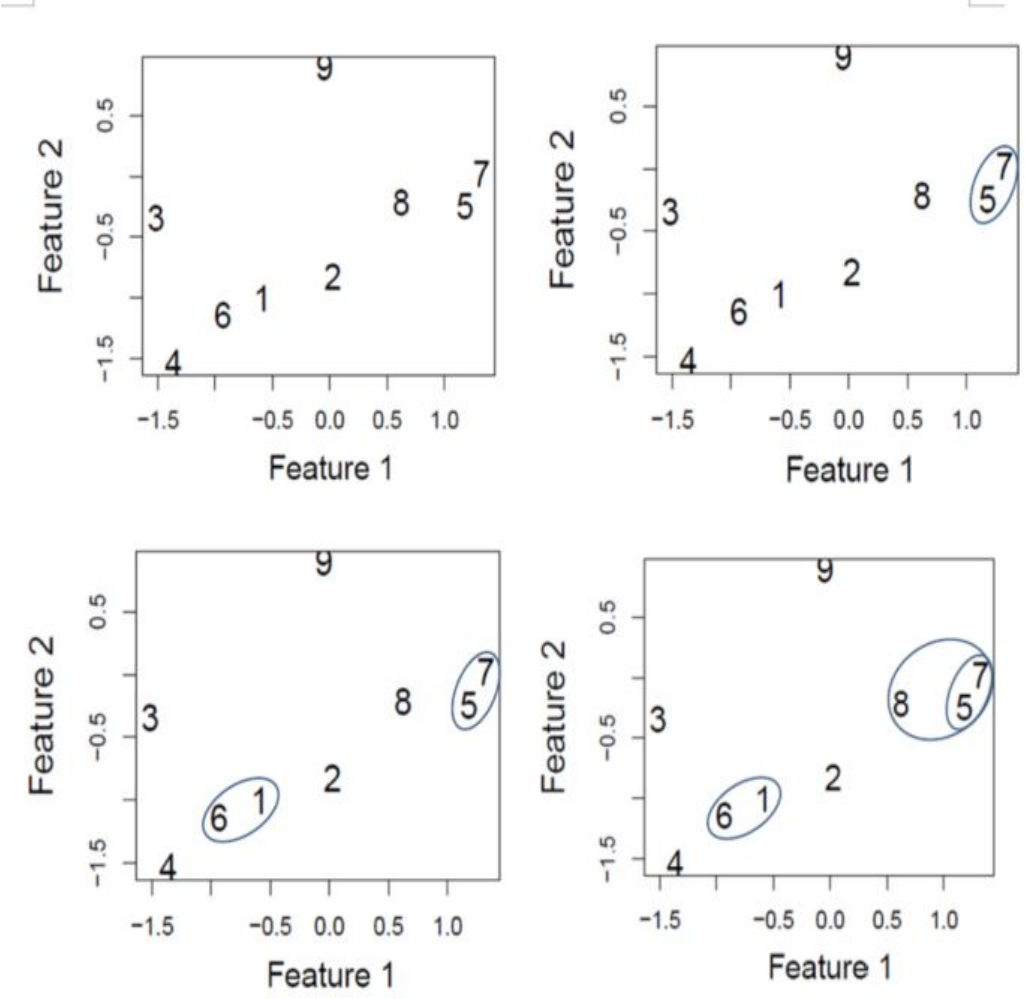
For $i = n, n-1, \ldots, 2$:

2. Fuse the two clusters with the *least dissimilarity* into one cluster (the dissimilarity will be the height in the dendogram where the fusion is placed). Once clusters have multiple points, one of the **linkage methods** will have to be used to define the dissimilarity between clusters(described in more detail later).

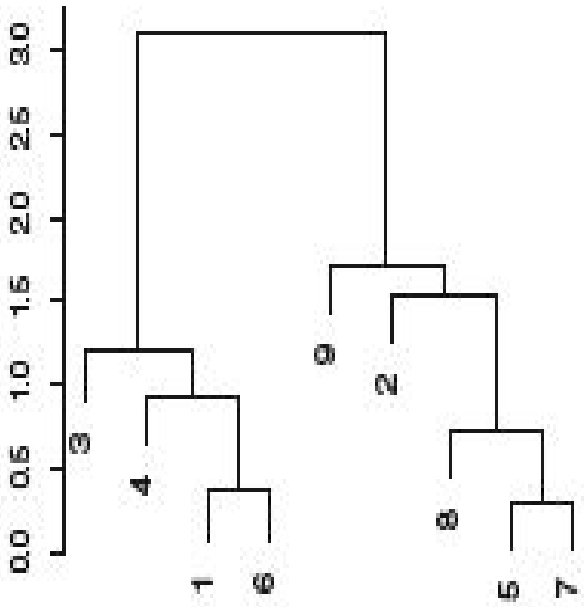
3. Re-compute the dissimilarity measures among the $i-1$ clusters and continue fusing them together until there are just 2 clusters.

Hierarchical Clustering Algorithm

An example of the algorithm in the previous slide (uses complete linkage):



…and so on until:


0.0
0.5
1.0
1.5
2.0
2.5
3.0
3
4
1
6
9
2
8
5
7

Linkage Methods

Complete Linkage - Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the **largest** of these dissimilarities.

Advantages: Gives comparable cluster sizes, less sensitive to outliers
Disadvantage: Cannot handle diverse shapes

Single Linkage - Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the **smallest** of these dissimilarities.

Advantage: Can handle diverse shapes
Disadvantages: Very sensitive to outliers/noise, and can create imbalanced or extended/trailing clusters

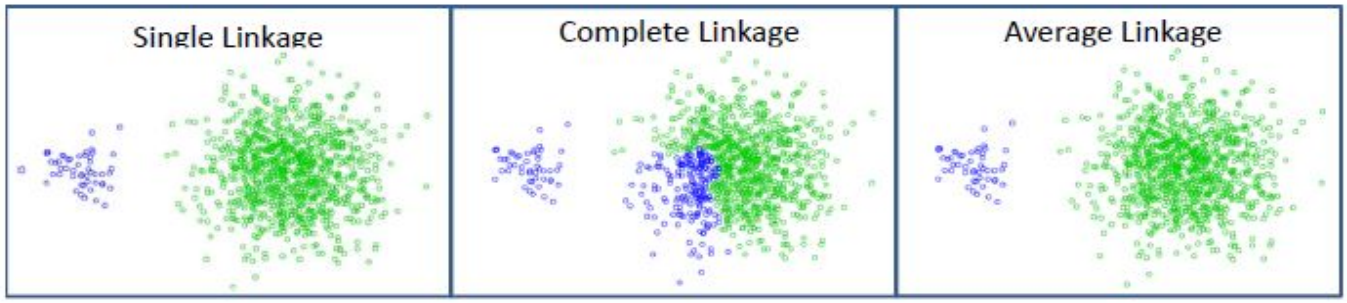
Average Linkage - Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the **average** of these dissimilarities.

This is a compromise of the above 2 methods, often works best with **spherical distributions**.

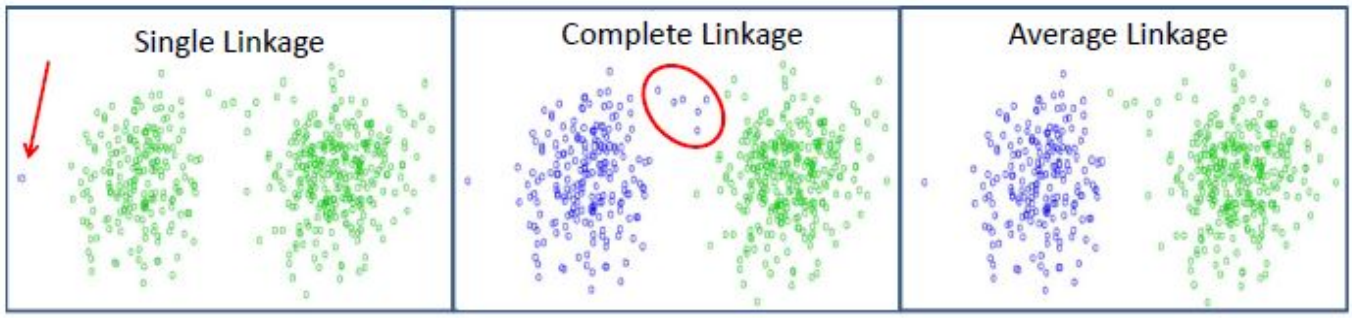
Complete and average linkage are the most commonly used methods in practice.

Linkage Methods Visualized

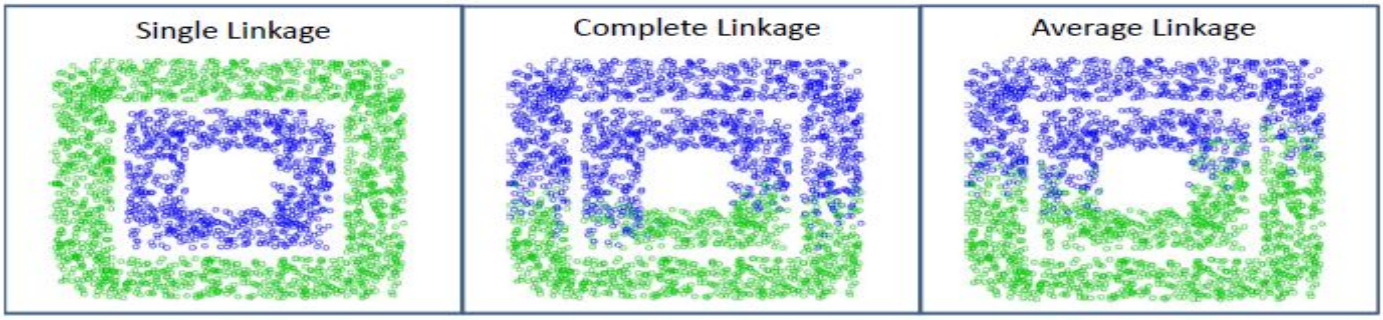
Unbalanced Clusters:



Outliers:

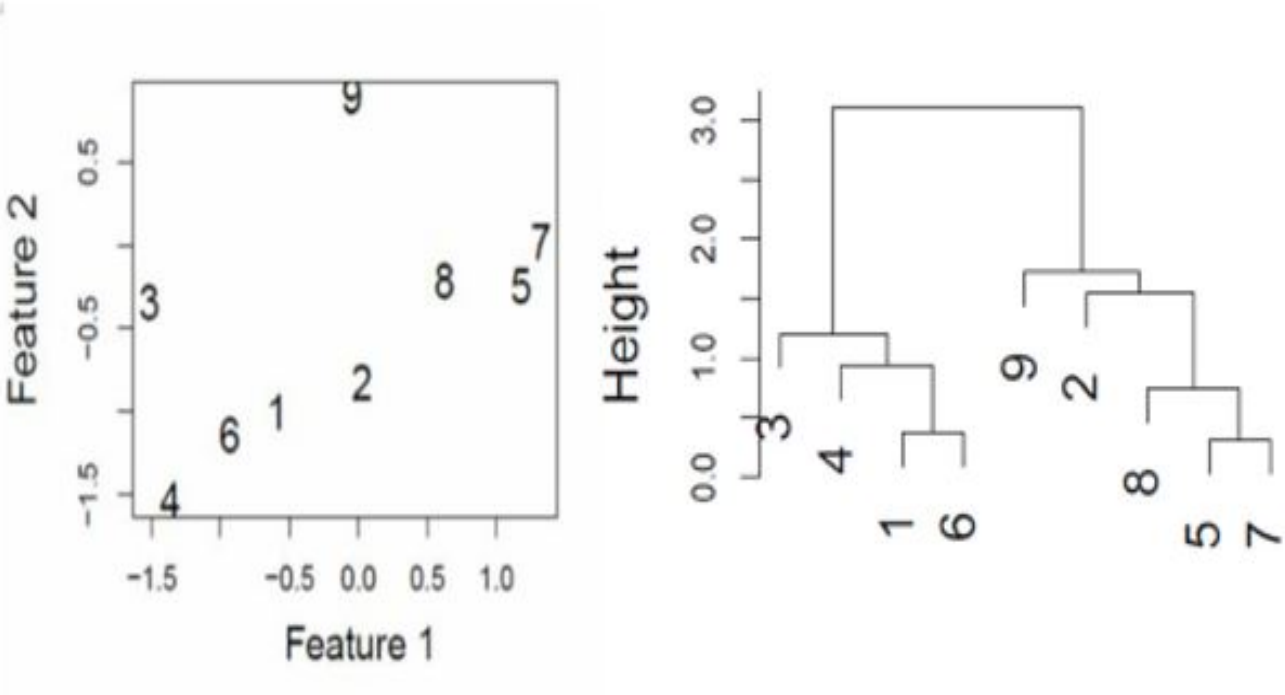


Non-spherical distribution/diverse shape:



Interpreting a Dendogram

Taking this graph and dendogram from the previous example:



The lower in the tree the fusions occur, the more similar the groups of observations are, whereas if fusions occur near the top of the tree, the observations are very different.

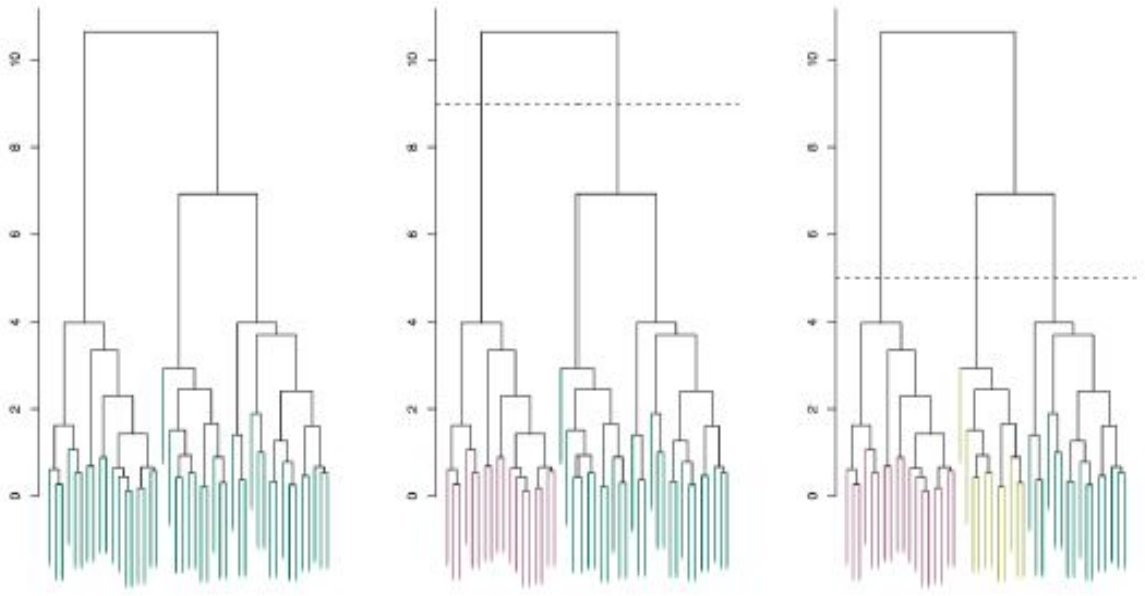
For example, 1, 6, and 4 are similar, 5,7, and 8 are similar, but those two groups are very different from one another.

The vertical axis shows the dissimilarity at the fusion. For example, point 4 and the cluster of (1,6) have dissimilarity of just under 1.0. The cluster of (3,4,1,6) and (9,2,8,5,7) have a dissimilarity of just above 3.0.

Important note: *One cannot conclude anything about two observations' similarity based on the horizontal axis.*

Choosing Hierarchical Clusters

To choose a solution from a dendogram, the data analyst must make a choice of at what dissimilarity to **cut** the tree. The number of clusters will vary based on this location, as shown below:



The dashed line is the location of the cut, with the second tree having 2 clusters and the third tree having 3.

K-means vs. Hierarchical

Hierarchical:
Advantages: Gives a family of possible solutions; computationally fast

Disadvantages: No optimization criterion; final solution chosen by the data analyst; different merging (splitting) criteria give different solutions

K-means:
Advantages: Computationally efficient, can be applied to high dimensional data, works well with equal-sized clusters.

Disadvantages: Results depend on initial cluster assignment, there may be empty/artificially small clusters.

No one method is better than the other. It is good to use both methods for the same data to explore various results.

Practice Exercises

10.2, 10.3, 10.10